Programming with VAX BASIC Graphics

Order Number: AI-HY14A-TE

This manual provides tutorial and reference material for VAX BASIC graphics.

Operating System and Version: VAX/VMS Version 4.5 or higher; MicroVMS Version 4.5 or higher

Software Version: VAX BASIC Version 3.0; VAX GKS Version 2.0

digital equipment corporation
maynard, massachusetts

August 1986

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a trademark of the American Mathematical Society.

ntents

CHAPTER 8 ADVANCED GRAPHICS PROGRAMMING TECHNIQUES

REFERENCE SECTION

CHAPTER 9 VAX BASIC GRAPHICS STATEMENTS

APPENDIX B DEVICE SPECIFICATIONS

:X

IRES

TABLES

eface

ınded Audience

This manual provides both tutorial and reference material on the graphics capabilities of VAX BASIC. Readers are presumed to be familiar with VAX BASIC programming techniques; however, no prior knowledge of graphics programming is required. This manual should be used with the other two manuals in the documentation set.

;ociated Documents

This manual is one of three manuals that form the VAX BASIC document set. The other two manuals are

VAX BASIC Reference Manual	Provides reference material and syntax for all VAX BASIC language elements except graphics capabilities
VAX BASIC User Manual	Provides tutorial material for VAX BASIC language constructs and information pertaining to programming with VAX BASIC on VAX/VMS systems

You may also be interested in the following supplementary manuals:

- *VAX BASIC Syntax Summary*
- *Introduction to BASIC*
- *BASIC for Beginners*
- *More BASIC for Beginners*

Structure of This Document

This manual has nine chapters and two appendixes.

The first eight chapters explain how to use the graphics capabilities o VAX BASIC. The remaining chapter consists of alphabetically arrange reference descriptions of each VAX BASIC graphics statement.

Programmers who have little or no experience using graphics should b with Chapter 1 and proceed with the chapters in sequence. Experienc graphics programmers may prefer to use the reference section (Chapte 9) and the chapters concerned with more advanced graphics techniqu (Chapters 6 through 8).

Chapter 1	Introduces you to VAX BASIC Graphics and the main categories of graphics statements
Chapter 2	Shows you how to put some simple graphics objects on y screen
Chapter 3	Shows you how to change the appearance of objects
Chapter 4	Explains how to change text attributes
Chapter 5	Explains the coordinate systems
Chapter 6	Describes how to create complex images
Chapter 7	Describes various ways of providing input for graphics
Chapter 8	Explains how to use multiple devices and device transforn tions
Chapter 9	Lists the rules and syntax for all VAX BASIC graphics statements
Appendix A	Describes how to call VAX GKS directly
Appendix B	Provides device-specific information

nventions Used in This Document

Convention	Meaning
`$ ASSIGN`	In command-line examples, the user's response to a prompt is printed in red; system prompts are printed in black.
. . .	A vertical ellipsis indicates that code which would normally be present is not shown.
UPPERCASE letters	Uppercase letters are used for VAX BASIC keywords and must be coded exactly as shown.
lowercase letters	Lowercase letters are used to indicate user-supplied names or characters.

Conventions used in the syntax diagrams in Chapter 9 are listed at the start of that chapter.

ummary of New and Changed Features ›r Version 3.0

Version 3.0 of VAX BASIC includes extensive graphics capabilities, structured error handling techniques, enhancements to file I/O and other new features. All of these features are documented in the *VAX BASIC Reference Manual* and the *VAX BASIC User Manual* except for the graphics features, which are documented in this manual. This section summarizes all of the major changes for this release.

Graphics Capabilities

VAX BASIC supports extensive graphics capabilities based on VAX GKS. The new graphics capabilities are available to you if you have the full or run-time VAX GKS kit installed on your system (Version 2.0 or later) and if you use supported graphics hardware. The main features of VAX BASIC graphics are as follows:

- A short learning period
- Convenient default values for attributes
- Statements consisting of English words in simple constructs
- Window and viewport settings that are easy to alter
- Graphics subprograms that can be invoked with a variety of transformation functions
- Input statements for interactive graphics programs
- Programs that can run on multiple devices
- Programs that run on any hardware supported by VAX GKS

Structured Error Handling

VAX BASIC supports structured error handling with WHEN ERROR constructs. When an error occurs during execution of statements in a protected block of code, the error is handled by the associated attached or detached handler. The following new statements and functions enhanc error-handling capabilities:

- WHEN ERROR
- RETRY
- CONTINUE
- HANDLER, EXIT HANDLER, and END HANDLER
- OPTION HANDLE
- CAUSE ERROR
- RMSSTATUS and VMSSTATUS

Although the new WHEN ERROR constructs are the preferred method for error handling, ON ERROR statements are supported for compatibili with previous versions of BASIC.

Optional Line Numbers

Line numbers are no longer required in VAX BASIC programs. A VAX BASIC program can have no line numbers at all, or it can use the traditional line-numbered statements; both are valid. A program with a line number on the first nonblank line is treated as a line-numbered prograr by the compiler. In the BASIC environment, programs with no line nun bers must be created with a text editor or copied into the environment with the OLD command.

Array Bounds

You can now specify the lower bound for any or all dimensions of nonvirtual arrays. Previously, VAX BASIC arrays could only be zero-based. In addition, two new functions, LBOUND and UBOUND, allow you to retrieve the lower and upper bounds of array dimensions.

Improvements for Procedure Invocations

This version of VAX BASIC includes additional flexibility for procedure invocations:

- If an external function is called as a procedure, VAX BASIC perform parameter validation exactly as if the declared function had been invoked as a function.
- The new keywords ANY and OPTIONAL ease parameter passing tc non-BASIC routines.

- Additional functionality has been added to the LOC function so that the address of an external function can be accessed.

PRINT USING Format Strings

Constant PRINT USING format strings are precompiled at compile time. Significant run-time performance gains can be achieved by recompiling programs that use constant format strings.

Single Keystroke Input

The new function INKEY$ allows you to detect a single keystroke typed at a terminal. Function and keypad keys return a descriptive text string, for example, "F17", and control characters return a single ASCII code.

I/O Enhancements

The following features have been added to enhance I/O capabilities:

- STREAM files are accessible with the OPEN statement.
- A WAIT clause can be added to the GET and FIND statements. This clause instructs VAX BASIC to wait on locked records rather than immediately returning the error RECBUCLOC (ERR = 154).
- The new keywords NX (next) and NXEQ (next or equal to) are synonyms for GT and GE respectively. These keywords make the GET and FIND statements more meaningful if an indexed file is accessed with descending keys.

Miscellaneous Features

- The new PROGRAM statement allows you to optionally name a main program unit. This name becomes the module name of the compiled source.
- You can return a procedure value or the status of an image upon exiting with the following statements:
  - END/EXIT PROGRAM
  - END/EXIT FUNCTION
  - END/EXIT DEF
- By default, VAX BASIC calls VAX EDT from the environment. The user or the system manager can select callable VAX EDT, the VAX Text Processing Utility (VAXTPU), or the VAX Language Sensitive Editor as the default editor. Start-up time for editing files within the environment is shorter as it is no longer necessary to spawn a subprocess to access editors that are callable.

- System managers can prevent users escaping to DCL level from the environment by setting the user's subprocess limit (PRCLM) to zero. A subprocess limit of 1 was previously required so that a user could use an editor within the environment.
- New extensions to the OPTION statement include the following:
  - OPTION ANGLE = *degrees-or-radians*
  - OPTION HANDLE = *severity-level*
  - OPTION CONSTANT TYPE = *data-type*
  - OPTION OLD VERSION = CDD
- The MID$ function can now be on the left side of an assignment statement. This feature allows partial string replacement.
- VAX BASIC statements, compiler directives, labels, and comment line can now start in column 1.
- You can include files from a text library with the %INCLUDE directive.
- The suffixes $ (for strings) and % (for integers) are allowed on explic-itly declared variables and constants.
- Extensions to the REMAP and MAP DYNAMIC statements allow you to redefine the storage allocated to a previously declared static string variable.
- New functions MAX and MIN are provided for the comparison of a series of arguments.
- The new MOD function divides one numeric argument by another and returns the remainder.
- The new compiler directive %PRINT allows you to print a message during the compilation of a source program without aborting the compilation.
- The new lexical directive %DECLARED allows you to determine whether or not a lexical variable has been declared.

Chapter 1

Introduction to VAX BASIC Graphics

VAX BASIC graphics statements have been designed so that you can create and manipulate pictures and incorporate graphics into your applications programs.

.1 Overview

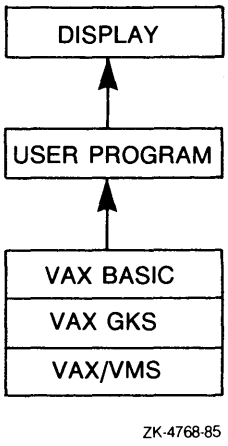
With VAX BASIC graphics statements you can:

- Draw lines and geometric shapes
- Display points with various markers
- Display text in various fonts and sizes
- Fill a defined area with various patterns
- Alter window sizes
- Accept input interactively
- Create graphs and designs
- Create complex images with graphics subprograms

These graphics are applicable over a wide range of conventional graphics hardware. A VAX BASIC program containing graphics statements can be run on any system running VAX/VMS Version 4.5 or higher that has VAX GKS Version 2.0 or higher installed.

VAX GKS provides extensive two-dimensional graphics capabilities; VAX BASIC provides an interface between your program and VAX GKS. The building blocks of this interface are illustrated in Figure 1–1. Single language statements allow you to easily access the capabilities of VAX GKS from within a VAX BASIC program.

Figure 1–1: Building Blocks of VAX BASIC Graphics



1.2 Before You Begin

The examples in this manual will run on any hardware supported by VAX GKS (although some examples may require changes for a particulaı terminal). Devices supported by VAX GKS include VT125 and VT240 terminals, both monochrome and color, and VAXstations I, II, and II/GP. Each of these devices has a designated number that VAX GKS recognize: and associates with the actual device.

Before writing VAX BASIC graphics programs, you should inform VAX GKS of your terminal type by assigning the designated number for your device to the logical name GKS$WSTYPE. You do this with the DCL command ASSIGN. The following command informs VAX GKS that you are using a VT240 terminal with color capabilities (device type 13).

```
$ ASSIGN 13 GKS$WSTYPE
```

If you do not assign the appropriate value to GKS$WSTYPE, VAX BASIC uses the system default device type (if one has been assigned). To find out which device type has been assigned to GKS$WSTYPE, enter the following command:

```
$ SHOW LOGICAL GKS$WSTYPE
```

If no translation is available for GKS$WSTYPE, VAX BASIC assumes that you have a monochrome VT240 (device type 14). A complete list of supported devices is provided in Chapter 9 under the OPEN...FOR GRAPHICS statement. Once you have established your terminal type, you should assign this value to the logical name GKS$WSTYPE in your login command procedure.

The quality of the generated image varies according to the display device that you use. The larger the number of pixels, the higher the resolution of the graphics image. Similarly, the quality of color in the images produced by VAX BASIC graphics programs is limited by the quality of the display device you use.

.3 VAX BASIC Graphics Statements

VAX BASIC graphics statements consist of English words (such as SET, ASK, and PLOT) in simple constructs. The statements are therefore easy to remember and comprehend. For example, the following statement accepts the coordinates of a single point from the keyboard:

```
LOCATE POINT x,y
```

The following statement increases the height of the text to a value of 0.05:

```
SET TEXT HEIGHT 0.05
```

VAX BASIC graphics statements can be grouped into the seven main categories listed in Table 1–1.

Table 1–1: Categories of VAX BASIC Graphics Statements

Statement Category	Purpose
Output statements	Draw graphics images such as points, lines, an areas, and also write text.
Attribute statements	Specify the exact appearance of a graphics object, for example, the style, size, thickness, and color of a line.
Picture statements	Define an output display in a fashion similar to VAX BASIC subprograms. When drawn, pictures can be shifted, scaled, sheared, and rotated.
Input statements	Accept input from various hardware devices such as a mouse or a keyboard.
ASK statements	Retrieve the current value of an attribute. The current value is assigned to a variable you supply.
Coordinate statements	Provide you with a means of expressing the relative position of an object and how much of a display should be shown, and allow you to limit the display to a particular area on the display surface. These statements ultimately affect the visible output on the display screen.
Control statements	Manage the opening and closing of VAX GKS, the initialization and selection of devices, and the clearing of display surfaces.

A typical VAX BASIC graphics program includes statements from several of these groups. The following sections provide more information on eacl group of statements.

.3.1 Output Statements

VAX BASIC provides several statements for drawing objects, including GRAPH POINTS, GRAPH LINES, GRAPH AREA, and GRAPH TEXT. You must supply VAX BASIC with information about where to place graphics objects on the display surface. You can do this with a simple coordinate system. For details about coordinate systems, see Section 1.3.6 and Chapter 5.

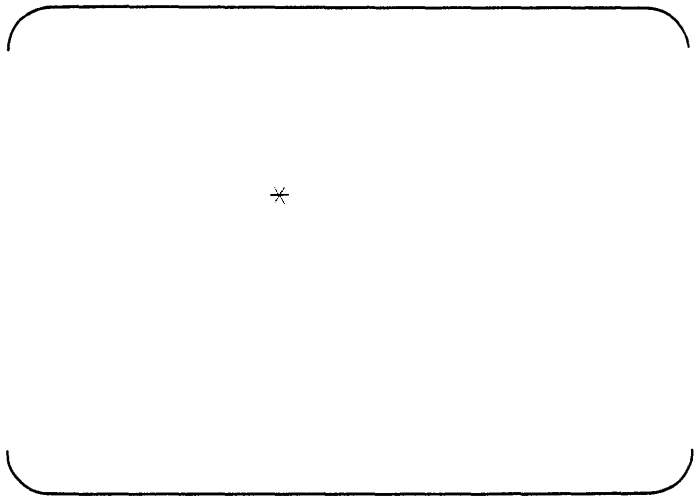
You separate the coordinates of a single point with a comma. Pairs of coordinates must be separated by a semicolon. Note that you cannot use parentheses to enclose the coordinates of a point as is customary in other representations of points; VAX BASIC signals an error if you do so.

The following statement marks a single point x,y with an asterisk on the display surface:

Example

```
GRAPH POINTS x,y
```

Output



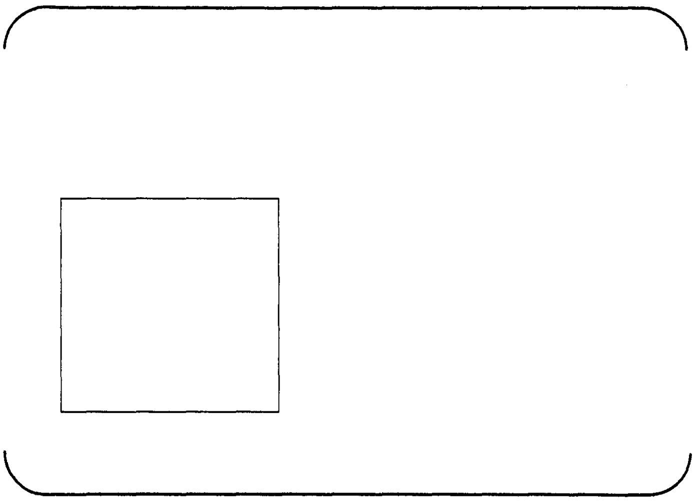
ZK-4862-85

The following GRAPH statement draws a line from the first to the secon
point, the second to the third point, and so on to complete a square:

Example

```
GRAPH LINES 0,0; 0,0.5; 0.5,0.5; 0.5,0; 0,0
```

Output



ZK-4861-85

PLOT statements provide additional capabilities for output. Chapters 2 and 6 discuss how to use the GRAPH and PLOT statements.

1.3.2 Attribute Statements

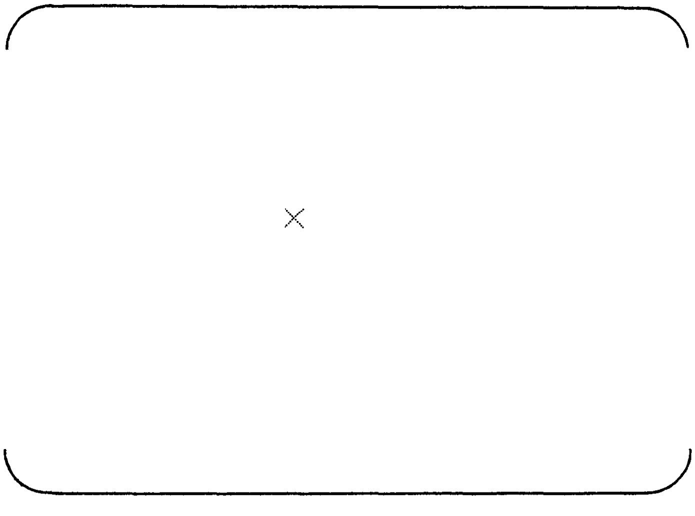
A graphics object has several *attributes*. For example, style, size, thickness and color are all attributes of a line. Each attribute has an initial default value; for instance, the style of a line is solid at the start of program execution. You can change the default style to produce a line of dots or dashes. Unless you explicitly change specific attributes, graphics objects are displayed with the default values.

The SET statement allows you to assign the attributes of your choice to a graphics object from within a program. For instance, in the previous GRAPH POINTS example, the point displayed on the screen is of the default style, an asterisk. Instead of an asterisk, you can specify that a point be displayed with one of several alternative marker styles. In the following example, the value 5 as an argument to the SET POINT STYLE statement produces a diagonal cross as the marker for the point.

Example

```
SET POINT STYLE 5
GRAPH POINTS x,y
    .
    .
    .
```

Output


ZK-4860-85

Similarly, you can change the text font so that characters are displayed with a hardware font rather than the default software font. To do this, you can include the following statement before drawing text:

```
SET TEXT FONT -1, "CHAR"
```

Chapters 3 and 4 describe the possible attributes for graphics objects and explain how to select them.

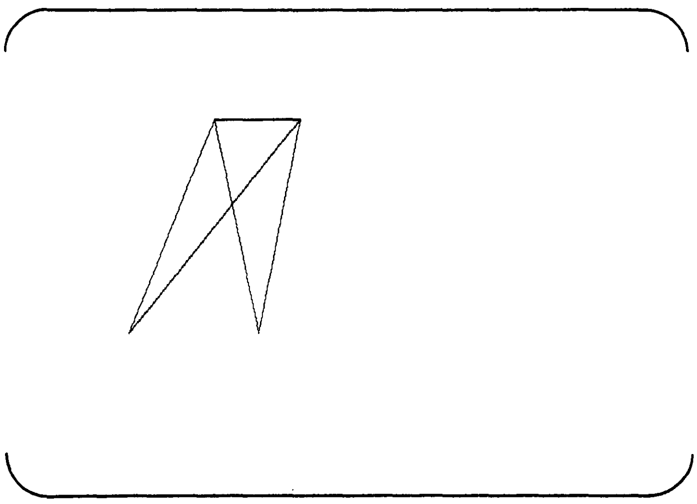
1.3.3 Picture Statements

A routine for drawing a particular image can be defined in a *picture,* a graphics subprogram. You can subsequently draw the picture and alter the original image by shifting, scaling, rotating, or shearing. Pictures can also be drawn recursively. You invoke a graphics picture with the DRAW statement. In the following example a triangle is defined in a picture and later invoked with the DRAW statement:

Example

```
PICTURE Triangle(SINGLE x,y)
   !+
   !Draw triangle with varying third point
   !-
   GRAPH LINES 0.3,0.7; 0.5,0.7; x,y; 0.3,0.7
END PICTURE

PROGRAM Triangle_demo
   EXTERNAL PICTURE triangle(SINGLE,SINGLE)
   !+
   ! Invoke the picture naming a third point
   !
   DRAW Triangle(0.4,0.2)
   !+
   ! Invoke the picture with a different third point
   !-
   DRAW Triangle(0.1,0.2)
END PROGRAM
```

Output


ZK-5405-86

Chapter 6 discusses graphics pictures; it explains parameter passing, picture definitions, and how to make changes in the picture image.

Two libraries of pictures are included with your VAX BASIC kit: a text library (BASIC$GRAPHICS.TLB) and an object library (BASIC$GRAPHICS.OLB). These libraries contain useful routines that you can use in your own programs. Both of these libraries reside in SYS$SYSROOT:[SYSHLP.EXAMPLES.BASIC]. Files from the object library can be linked with main programs; source files from the text library can be included in your programs with the %INCLUDE directive.

1.3.4 Input Statements

VAX BASIC graphics programs accept data from an input device in a fashion similar to the INPUT and MAT INPUT statements. However, graphics data is accepted by the LOCATE and MAT LOCATE statements. In addition to entering the data as characters on a keyboard, a user can supply the data by positioning a mouse, by positioning a cursor on a menu selection, or by other means.

SET statements allow you to set up the screen display presented to a user. For instance, the following example shows how to set up a list of menu choices to be presented to a user. A subsequent LOCATE CHOICE statement displays the menu and accepts the user's selection.

Example

```
SET INITIAL CHOICE                    &
            , LIST ("view"            &
                  ,"update"           &
                  ,"delete"           &
                  ,"quit")            &
                  : 1
LOCATE CHOICE to_do
```

Chapter 7 shows you how to use the graphics input statements.

1.3.5 ASK Statements

At any point in your program you can ask for the current value of an attribute. You request the value with an ASK statement, and the value returned is assigned to a variable you supply. The ASK statement retrieves the current value of the attribute, that is, the value that would apply if an output statement were performed at that time. Values for common attributes are provided in the appropriate chapters. The following statements are examples of ASK statements. Note that in graphics programs, floating-point variables should be declared as SINGLE, and integers should be declared as LONG.

Example

```
DECLARE LONG what_color,which_style,point_size,        &
        SINGLE how_tall

ASK TEXT COLOR what_color
ASK TEXT HEIGHT how_tall
ASK LINE STYLE which_style
ASK POINT SIZE point_size
```

Within a program, you can ask for current attribute values, change the attribute values, and later use the stored values to restore the previous settings. The ASK statements are also useful for determining the particular capabilities of the devices that you use and adapting your programs accordingly. The ASK statements are described in various chapters throughout the manual.

3.6 Coordinate Statements

You must supply VAX BASIC with information about where to place the graphics object on the display surface. To supply this information in terms of pixels on the screen would be tedious. Instead, you can supply the information by using a simple coordinate system. You express a point in terms of its relative position from specified x- and y-axes on your display surface. This way, any point on your screen can be described by a pair of coordinates. VAX BASIC transforms your coordinates into the actual coordinates for a supported device automatically so that you do not have to write various versions of your program to accurately position the display on a specific output device.

VAX BASIC provides statements that allow you to manipulate the display of a graphics object. You can select just a section of the screen for a display, you can overlay one image with another, or you can build up one large image from several smaller images. Chapters 5, 6, and 8 discuss how to control the positioning of images on an output device.

1.3.7 Control Statements

Control statements add flexibility to your programs. They allow you to use an alternate output device, or even use several devices at one time. Statements such as the following allow you to control when and where display from your program is activated.

Example

```
OPEN "VT70:" FOR GRAPHICS AS DEVICE #2
    .
    .
    .
CLOSE DEVICE #2
    .
    .
    .
```

When VAX BASIC opens a device for graphics, the screen is automaticall cleared.

Unless you plan to use a device other than the terminal at which you enter the program, you need not be concerned with these statements. Chapter 8 provides details on how to use control statements.

1.4 Using VAX BASIC Defaults

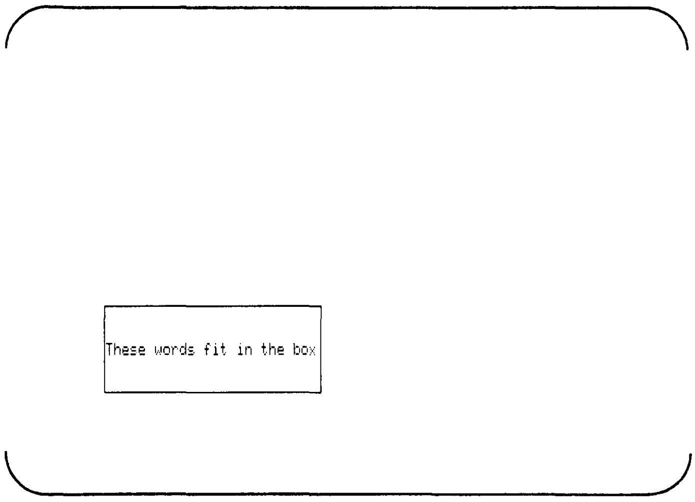
For most purposes, VAX BASIC provides you with useful default values. If you are programming a display to appear on your current terminal, foı instance, you need not supply a device identification number; unless you state otherwise, VAX BASIC assumes that you are referring to your curreı device. Similarly, a square region of your screen is used as the default area for your display unless you specify a different section as your outpu area. It follows that you can keep your graphics programs simple when you wish to.

The following example shows a simple graphics program that uses only the default values. The program draws a box on the terminal screen and then places text inside the box. The built-in VAX BASIC defaults are use for the style and thickness of the line, the text font and size, the device, and the amount of the screen used for the display.

Example

```
PROGRAM Sample
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE CONSTANT x = 0.1                    &
                        ,y = 0.1
  !+
  ! Connect four points to form box
  !-
  PLOT LINES x,y; x,y+0.2; x+0.65,y+0.2; x+0.65,y; x,y
  !+
  ! Place the text in the box
  !-
  GRAPH TEXT AT x, y + 0.08 : "These words fit in the box"
END PROGRAM
```

Output



ZK-4859-85

When you do require values other than the defaults, you can change the defaults by supplying the appropriate values with the SET statement.

1.5 Program Development Options

As with other VAX BASIC programs, you can develop a graphics prograr within the BASIC environment as well as directly at DCL level. Executabl graphics statements can be run in immediate mode. Statements that conflict with compiler commands (such as the SET statement) can be entered in immediate mode provided that each statement is preceded wit a backslash (\).

VAX BASIC graphics capabilities are fully integrated with the VAX/VMS program development tools. For instance, the optional VAX Language-Sensitive Editor includes extensive language definitions for VAX BASIC graphics statements. Similarly, you can locate run-time errors in a VAX BASIC graphics program with the VAX/VMS Debugger.

Error messages related to graphics are listed in the *VAX BASIC User Manual* along with all other error messages for VAX BASIC.

1.6 Graphics Standards

The Graphical Kernel System (GKS) has been adopted as the Internation‹ Standard by the International Organization for Standardization (ISO) and the American standards body, ANSI. The main objective of GKS is to create and manipulate graphics images independently of the computer or device being used.

The VAX BASIC graphics package is based on DIGITAL's implementatior of GKS on VAX/VMS systems, VAX GKS. Many VAX BASIC graphics statements correspond directly to one or more of the VAX GKS routines described in the VAX GKS documentation. The VAX BASIC graphics statements provide many of the capabilities described in VAX GKS and, therefore, many of the capabilities prescribed by the ISO.

Chapter 2

Displaying Graphics Objects

This chapter shows you how to use the graphics statements GRAPH, PLOT, MAT GRAPH, and MAT PLOT to display points, lines, areas, and text on your screen.[1]

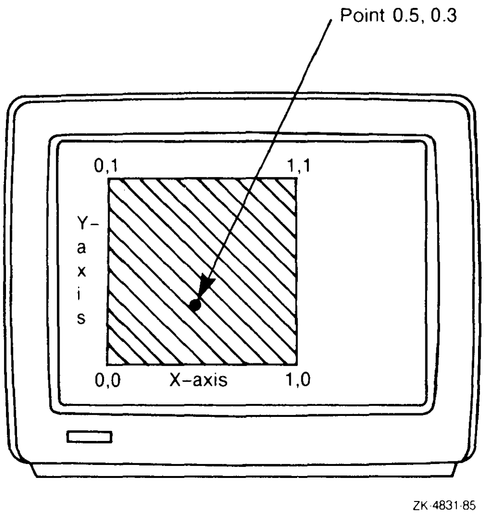
1 The Default Drawing Board

Before you can direct VAX BASIC to display a point on your terminal screen, you have to consider where the point should be and how you can specify that position. To specify the position of a point, you use a *coordinate system*. The coordinate system you use to specify a point to VAX BASIC is the Cartesian system; it consists of a two-dimensional coordinate plane (referred to here as the *default drawing board*).

The default drawing board is usually a square region of your terminal screen, although this is dependent on the particular device you use. For instance, if you have a VT125 or VT240 terminal, the default drawing board fills the largest square region that it can, starting with the lower left corner of the screen. Figure 2–1 illustrates the default drawing board.

[1] On VAXstations, output is displayed on a new window and the output display disappears when the new window is closed. There are various ways to keep the output display active; for instance, you can include SLEEP statements in your programs.

Figure 2–1: The Default Drawing Board on a VT125 Termin



The coordinate system consists of two perpendicular axes with an origin zero. The *x-axis* extends to infinity, increasing positively to the right anc negatively to the left. The x-axis is the horizontal axis in the coordinate plane: the default x-axis extends along a section of the base of your screen. The *y-axis* also extends to infinity, but increases positively upwaı and negatively downward. The y-axis is the vertical axis: the default y-axis bounds your screen on the extreme left.

With Cartesian coordinates, you define the position of a point by its distance from the x- and y-axes. The *x-coordinate* of a point is the horizontı distance from the vertical axis to the point. The *y-coordinate* of a point is the vertical distance from the horizontal axis to the point. When a point specified, the coordinates are separated by a comma, for instance *x,y*.

NOTE

Coordinates of a single point must be separated by a comma. Pairs of coordinates must be separated by a semicolon. You cannot use parentheses to enclose the coordinates of a point as is customary in other representations of points; VAX BASIC signals an error if you do so. However, parentheses are allowed when expressions are supplied as coordinate values. The following examples show valid syntax for the coordinates for a single point:

- 0.5,0.8
- x,y
- 0.1,(0.1 + y)
- 1/5,(2/7)

Distance is the abstract measurement of the length of each axis. For the default drawing board, the distance is specified on a scale of 0 to 1. Each axis begins at 0 at the bottom left corner of your screen and extends to 1 at the end of each axis. (Note that the axes actually extend to infinity; the limits of 0 through 1 are imposed only for the default drawing board.) For the default drawing board, the coordinate pair 0,0 specifies the point of intersection between the two axes at the lower left corner of your screen. The coordinate pair 0.5,0.5 specifies a point in the center of the drawing board. Figure 2–1 illustrates the specifications for the point 0.5,0.3 on the default drawing board of a VT125 screen.

Throughout this manual the variables *x* and *y* represent the x- and y-coordinates of the point on your screen. Notice that any objects plotted with negative coordinate values, or values greater than 1, are beyond the scope of the default drawing board and therefore are not displayed on your screen. For instance, the points 5,7 and -2,3 are not displayed on the default drawing board. You can supply a value from 0 through 1 with up to six digits of precision; however, the precision in a display is dependent on the quality of the graphics output device you use.

For more information about coordinates and how to change the default drawing board with the SET WINDOW statement, see Chapter 5. Chapter 5 shows you how to alter the drawing board so that the coordinates of a point can have values of less than 0 and more than 1. The examples in this chapter and Chapters 3 and 4 are limited to the default.

2.2 Displaying Points

The GRAPH POINTS statement displays one or more points on your screen. To display points, GRAPH POINTS must be followed by one coordinate pair for each point. To display a single point, supply one paiı of coordinates, for example:

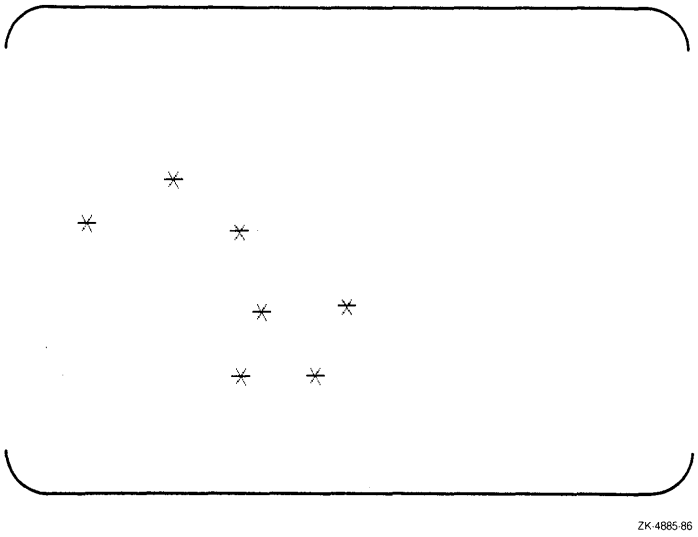
```
GRAPH POINTS 0.1,0.5
```

GRAPH POINTS marks the point with the default point style, an asterisl (*). To display several points, separate each coordinate pair with a semicolon (;).

Example

```
GRAPH POINTS  0.1,0.5;     &
              0.3,0.6;     &
              0.45,0.475;  &
              0.5,0.29;    &
              0.45,0.14;   &
              0.625,0.14;  &
              0.7,0.3
```

Output



Displaying Lines

The GRAPH LINES statement displays one or more lines on your screen. You must provide both a starting point and an ending point for the line. Therefore, you must supply GRAPH LINES with the coordinates for at least two points. To draw one line, supply GRAPH LINES with two coordinate pairs separated by a semicolon (;).

```
GRAPH LINES 0.8,0.2; 0.7,0.8
```

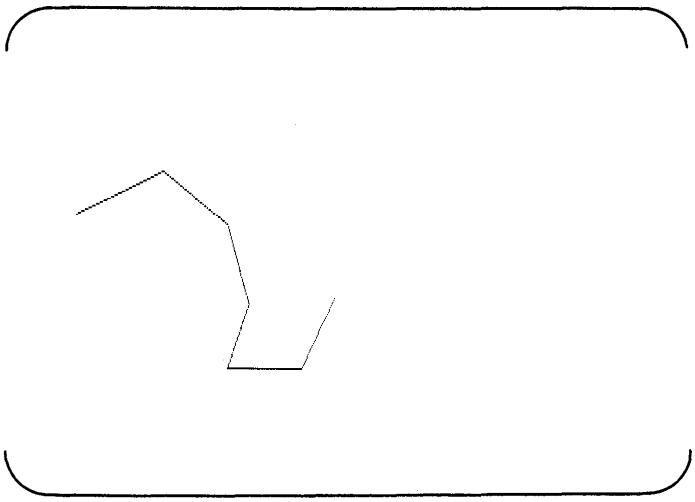
GRAPH LINES connects these two points with the default line style, a solid line.

When you supply several coordinate pairs as parameters, GRAPH LINES draws a line from the first point to the second point, from the second point to the third, from the third to the fourth, and so on.

Example

```
GRAPH LINES 0.1,0.5;     &
            0.3,0.6;     &
            0.45,0.475;  &
            0.5,0.29;    &
            0.45,0.14;   &
            0.625,0.14;  &
            0.7,0.3
```

Output



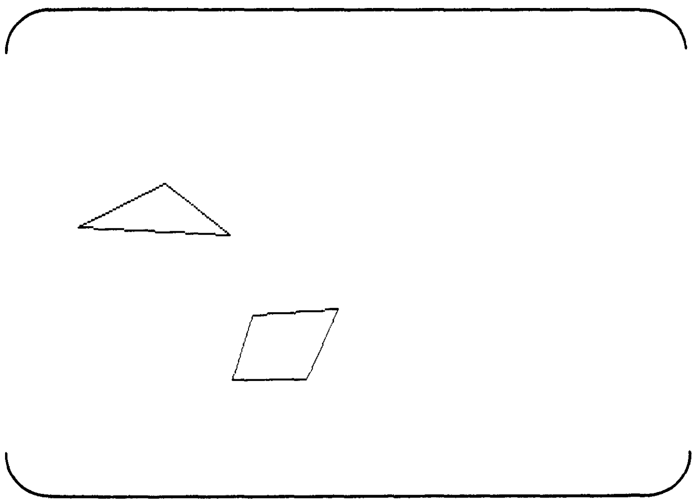
ZK-4886-86

You can also use GRAPH LINES to draw closed shapes. The next examı
uses the same seven points and two GRAPH LINES statements to draw
quadrilateral and a triangle. In each case the first point must be supplie
again as the last point so that the closing line is drawn.

Example

```
!+
!Supply three points and repeat the first
!to enclose a triangle.
!-
GRAPH LINES 0.1,0.5;    &
            0.3,0.6;    &
            0.45,0.475; &
            0.1,0.5
!+
!Supply four points and repeat the first
!to enclose a quadrilateral
!-
GRAPH LINES 0.5,0.29;   &
            0.45,0.14;  &
            0.625,0.14; &
            0.7,0.3;    &
            0.5,0.29
```

Output

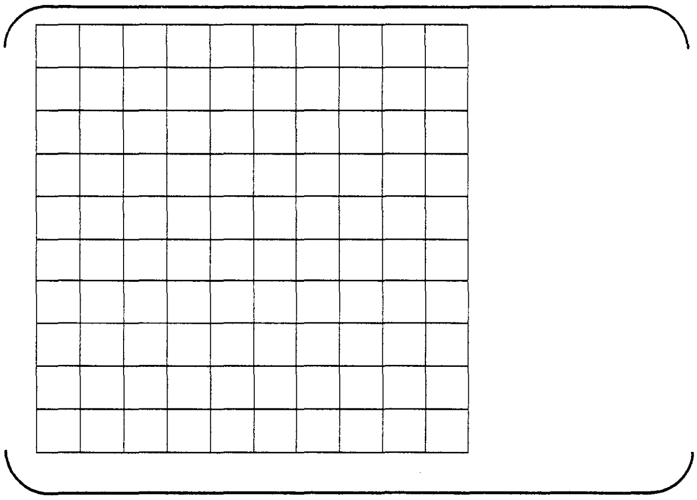


The following example shows the GRAPH LINES statement used within a FOR...NEXT loop to draw a 10 by 10 grid. When displayed on your screen, this grid covers the default drawing board. You can use such a grid to sketch drawings and to calculate the coordinates of points you want to use in your programs.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE counter
FOR Counter = 0 TO 1 STEP 0.1
    !+
    !Draw the horizontal lines
    !-
    GRAPH LINES Counter,0.0; Counter,1.0
    !+
    !Draw the vertical lines
    !-
    GRAPH LINES 0.0,Counter; 1.0,Counter
NEXT Counter
GRAPH LINES 0.0,1.0; 1.0,1.0; 1.0,0.0
END
```

Output



You can display lines with the PLOT LINES statement as well as the GRAPH LINES statement. However, PLOT LINES provides you with additional control over the display, as described in the following section.

.4 Controlling the Beam of Light

When you write with a pencil on paper, you often raise the pencil to move from one position to another. While the pencil is in the air, it leaves no mark or trail on the paper. When you press the pencil onto the paper, the pencil marks the paper with a trail or line.

A similar concept exists in computer graphics. In computer graphics you have the ability to switch a beam of light on or off. When the beam of light is switched off, it is possible to move a plotter from one position to another without leaving a trail or a line. When the beam of light is switched on, a plotter draws a line as it moves from one position to the next. In the case of graphics printers, the beam of light is analogous to a laser beam or ink jet.

The PLOT LINES statement gives you control over whether the beam of light is left on after drawing the specified line. To leave the beam on, put a semicolon at the end of the last point in a PLOT LINES statement. To switch the beam off, use a PLOT LINES statement with no points, or with no semicolon at the end of a point list. The following statement switches the beam of light on because it ends with a semicolon.

```
PLOT LINES 0.65,0.2;
```

The beam of light is on, but no line is plotted because only a starting point is specified. If another point is supplied in a further PLOT LINES statement, as in the next example, the beam of light will leave a trail from point 0.65,0.2 to the point in the next statement supplied, 0.3,0.75.

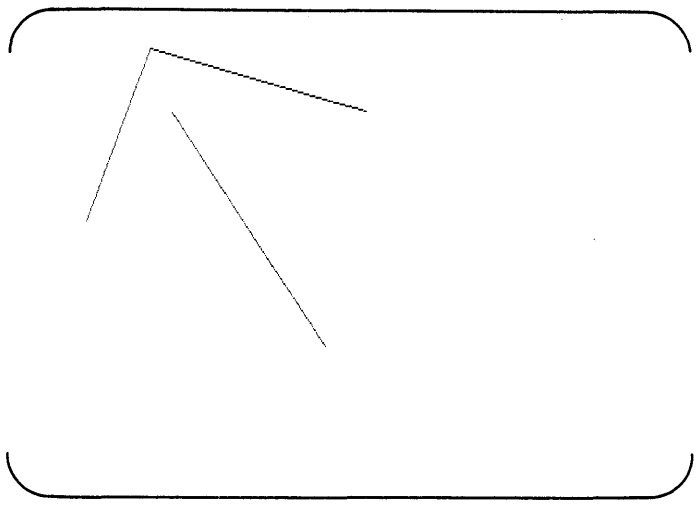
```
PLOT LINES 0.65,0.2;
PLOT LINES 0.3,0.75
```

After the point 0.3,0.75 has been reached, the beam is switched off because there is no semicolon at the end of the statement. A further PLOT LINES statement would not be connected to the point 0.3,0.75, as the beam is off. The positioning of the semicolons controls the light beam. Consider the output from the following example.

Example

```
PLOT LINES 0.65,0.2; !Leave the beam on
PLOT LINES 0.3,0.75  !Switch the beam off
PLOT LINES 0.1,0.5;  !Leave the beam on
PLOT LINES 0.25,0.9; 0.75,0.75 !Switch the beam off and display output
```

Output



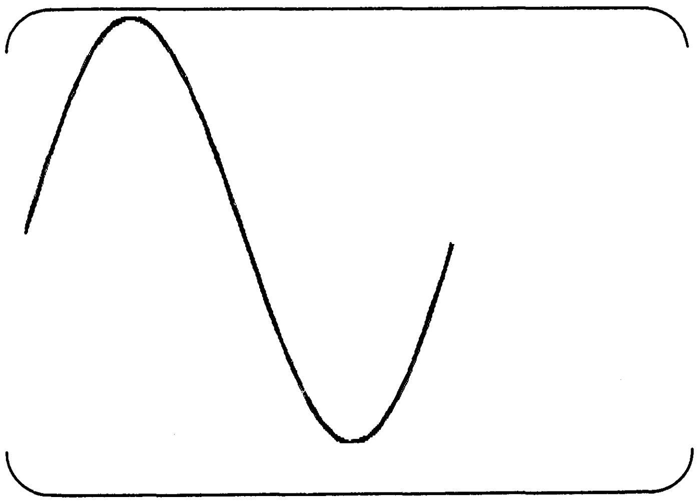
ZK-4888-86

The PLOT LINES statement can include coordinates for just one point only when the statement ends with a semicolon, or when the beam was already left on by a previous statement. This feature is useful when a sequence of points can be calculated in a formula and each new point supplied to the PLOT LINES statement in a program loop. The following example shows how the PLOT LINES statement can be used within a FOR...NEXT loop to draw an approximation of a sine curve. (Note that adjustments to the standard formula have been made to accommodate the default drawing board.)

Example

```
OPTION TYPE =  EXPLICIT
DECLARE SINGLE turn,x,y
!+
!Plots an approximation of a sine curve
!-
FOR turn = 0 TO 2 * PI STEP 0.1
    x = (turn/(2 * PI))
    y = ((SIN(turn) * 0.5) + 0.5)
    PLOT LINES x,y;
NEXT turn
!+
!Switch off the beam and display output
!-
PLOT LINES
END
```

Output



Each pass through the loop leaves the light beam on because of the semicolon at the end of the PLOT LINES statement. The final PLOT LINES statement outside the FOR...NEXT loop switches off the light beam as no points are supplied. You can also switch off the light beam with any other graphics statement, except another PLOT LINES statement with a semicolon.

Points connected with PLOT LINES statements that leave the beam on are not actually displayed one at a time; the output is stored until a graphics statement switches off the beam. When the beam is switched off, the output is displayed. In the previous example, the sine curve is not displayed until the PLOT LINES statement outside the loop is executed. If the beam is not switched off during program execution, VAX BASIC displays the output when the program terminates.

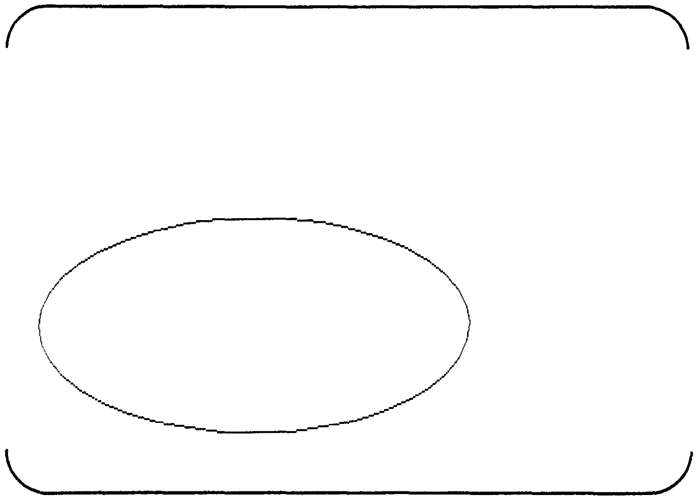
Note that the standard formula for a sine curve gives coordinate values of greater than 1 and less than 0; the sine curve would therefore extend beyond the boundaries of the default drawing board. Chapter 5 shows how to change the default drawing board and provides an example of displaying a full sine curve.

The following partial program uses the PLOT LINES statement to display an oval.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE turn, x, y,                  &
        LONG I
FOR I = 0 TO 40
   Turn  = 2 * PI * I/40
   x = (SIN(Turn) + 1)/2
   y = (COS(Turn) + 1)/4
   PLOT LINES x,y;
NEXT I
!+
!Switch off the beam
!-
PLOT LINES
END
```

Output



ZK-4889-86

Use the PLOT LINES statement when you need to control the beam of light in the display area or when you want to draw lines connecting points that are supplied only one at a time. Use the GRAPH LINES statement when you can supply at least two points at a time and when you do not need to control the beam of light. The PLOT statement has additional functions that this section has not addressed. For more information about the PLOT statement, see Chapter 6.

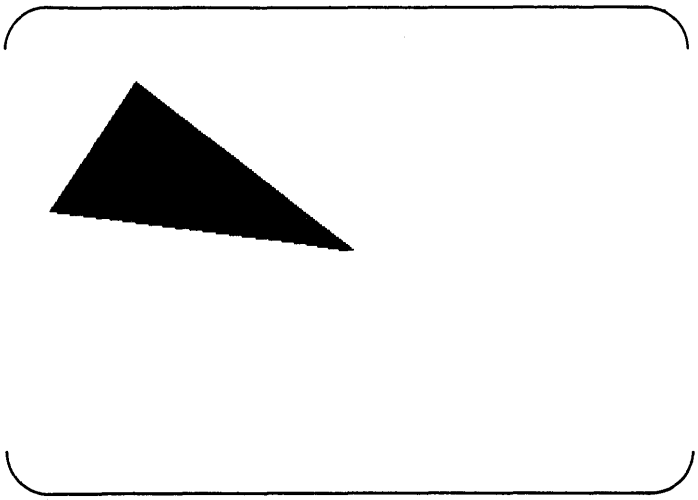
2.5 Displaying Areas

You can outline shapes with GRAPH LINES or PLOT LINES statements. VAX BASIC also provides you with a method of displaying the area of a shape. The GRAPH AREA statement allows you to fill the area of a shape. The default fill style for GRAPH AREA is solid.

Example

```
GRAPH AREA 0.3,0.8; &
           0.1,0.5; &
           0.8,0.4
```

Output


ZK-4891-86

Notice that the first point does not need to be repeated for the GRAPH AREA statement. The following two statements both produce a triangle: the GRAPH LINES statement draws the outline using four points, and the GRAPH AREA statement fills the area using three points.

Example

```
!+
!Only three points supplied for the area of a triangle
!-
GRAPH AREA 0.3,0.8; &
           0.1,0.5; &
           0.8,0.4
!+
!The first point must be repeated for the triangle outline
!-
GRAPH LINES 0.3,0.8; &
            0.1,0.5; &
            0.8,0.4; &
            0.3,0.8
```

The following statement displays the area of the default drawing board on your screen.

```
GRAPH AREA 0.0,0.0; 0.0,1.0; 1.0,1.0; 1.0,0.0
```

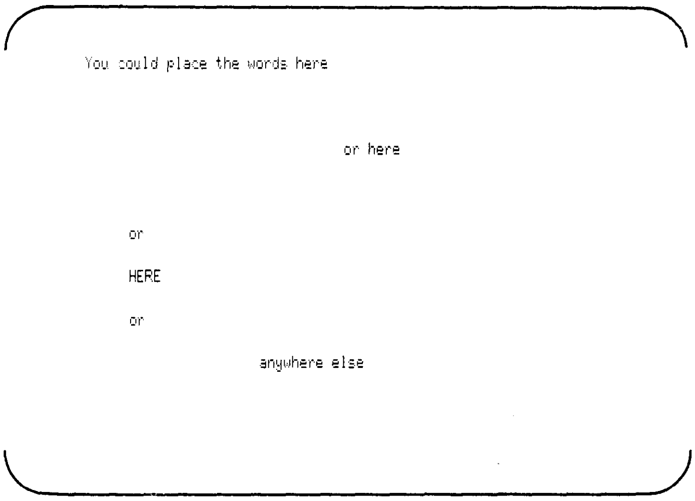
2.6 Displaying Text

You often need to position text on the screen as precisely as you position diagrams. The GRAPH TEXT statement allows you to supply the text you want displayed as well as the exact starting position. The starting position must be a coordinate pair. The text can be any string expression following a colon (:).

Example

```
DECLARE STRING CONSTANT selection = "or"
GRAPH TEXT AT 0.1,0.8 : "You could place the words here"
GRAPH TEXT AT 0.7,0.6 : selection + " here"
GRAPH TEXT AT 0.2,0.4 : selection
GRAPH TEXT AT 0.2,0.3 : "HERE"
GRAPH TEXT AT 0.2,0.2 : "or"
GRAPH TEXT AT 0.5,0.1 : "anywhere else"
```

Output



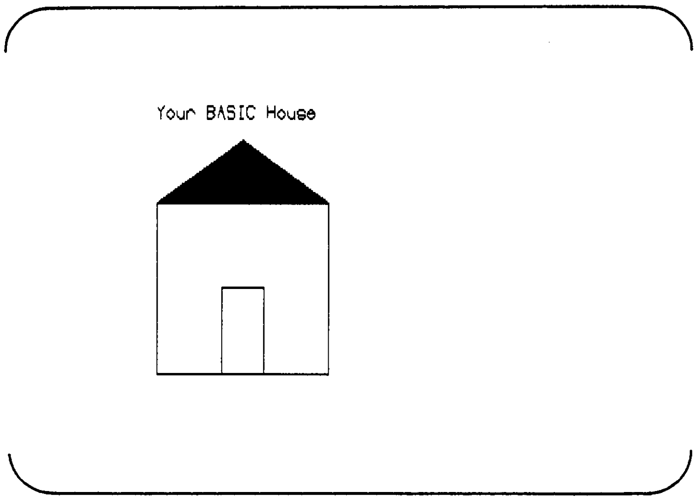
ZK-4864-85

You can easily label a diagram with the GRAPH TEXT statement, as shown here:

Example

```
GRAPH LINES 0.3,0.6; 0.7,0.6; 0.7,0.2; 0.3,0.2; 0.3,0.6
GRAPH LINES 0.45,0.2; 0.45,0.4; 0.55,0.4; 0.55,0.2
GRAPH AREA 0.3,0.6; 0.5,0.75; 0.7,0.6
GRAPH TEXT AT 0.3,0.8 : "Your BASIC House"
```

Output



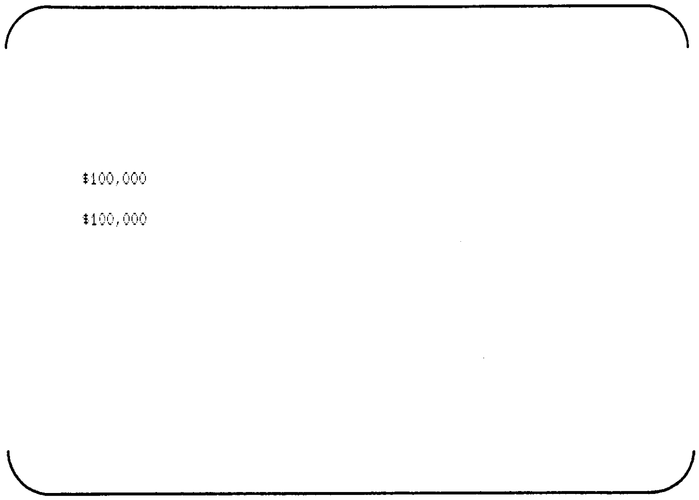
ZK-5505-86

You can use VAX BASIC built-in functions to format string or numeric expressions. For example, you can format a numeric expression with the function FORMAT$. Both of the following GRAPH TEXT statements display the same output:

Example

```
DECLARE STRING format_string
GRAPH TEXT AT 0.1,0.5 : FORMAT$(100000,"$$##,###")
    .
    .
    .
format_string = FORMAT$(100000,"$$##,###")
GRAPH TEXT AT 0.1,0.4 : format_string
```

Output



```
$100,000

$100,000
```

ZK-4867-85

!.7 Supplying Coordinates in Arrays

When you have many points to supply to a graphics statement, it can be more convenient to use arrays than to list each individual coordinate. VAX BASIC provides two types of graphics statements that display coordinates from arrays: MAT GRAPH and MAT PLOT. The MAT PLOT statements are discussed in Chapter 6. The following sections discuss the MAT GRAPH statements.

In a GRAPH statement, each point is represented by two coordinates: an x- and a y-coordinate. Similarly, in a MAT GRAPH statement each point is represented by two coordinates: an x-coordinate supplied in a one-dimensional array, and a y-coordinate supplied in another one-dimensional array. You supply the MAT GRAPH statement with two whole arrays: one array of x-coordinates and one array of y-coordinates. The first point is specified by the first element in the first array (*x-array*) as the x-coordinate, and the first element in the second array (*y-array*) as the y-coordinate, for example:

```
MAT GRAPH LINES x_array, y_array
```

The coordinates are taken,, one from each array, in sequence. Notice that you only specify the name of the array; you do not specify the array elements, nor do you specify the whole array with parentheses. Virtual arrays and packed decimal arrays are invalid.

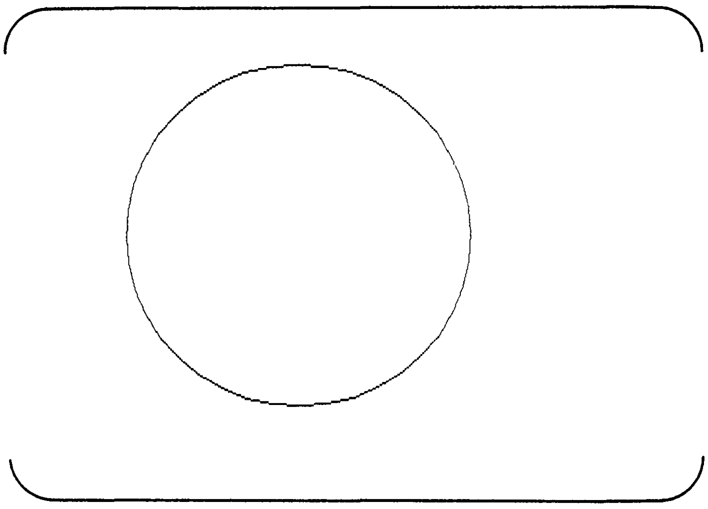
The following example displays a circle on the terminal screen using the MAT GRAPH LINES statement.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE CONSTANT radius = 0.4
DECLARE LONG CONSTANT npoints = 40
DECLARE SINGLE increment,              &
               x,y,turn,               &
        LONG Counter
DIM SINGLE x_array(npoints), y_array(npoints)
```

```
increment = 2 * PI/npoints
turn = 0
!+
!Fill the arrays
!-
FOR Counter = 0% TO npoints
      x = COS(turn) * radius
      y = SIN(turn) * radius
      !+
      !Adjust points for center of 0.5,0.5
      !-
      x_array(Counter) = x + 0.5
      y_array(Counter) = y + 0.5
      turn = turn + increment
NEXT Counter
!+
!Draw the lines connecting each point
!-
MAT GRAPH LINES x_array, y_array
END
```

Output



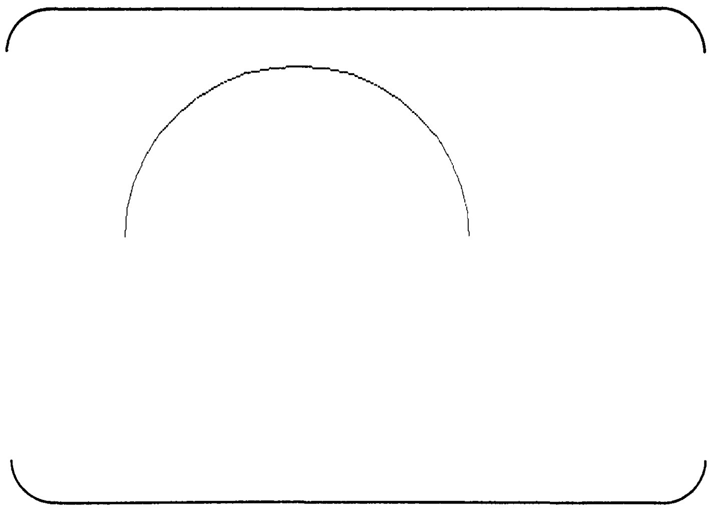
The MAT GRAPH LINES statement in the circle example can be replaced with a MAT GRAPH POINTS statement to display a circle of points. Alternatively, a MAT GRAPH AREA statement in this example would produce a solid circle.

The following program illustrates the use of the COUNT clause in a MAT GRAPH LINES statement. The COUNT clause allows you to terminate the display after the specified number of points has been processed, ignoring any remaining array elements. In the example only half of the points in the circle are specified by the index in the COUNT clause. There are actually 41 points in the circle; the MAT GRAPH LINES statement connects the first 21 points from the arrays and displays a semicircle.

Example

```
FOR Counter = 0% TO npoints
      x = COS(turn) * radius
      y = SIN(turn) * radius
      x_array(Counter) = x + 0.5
      y_array(Counter) = y + 0.5
      turn = turn + increment
NEXT Counter
!+
!Draw the lines connecting the first half of the points
!
MAT GRAPH LINES , COUNT (npoints/2 + 1) : x_array, y_array
END
```

Output

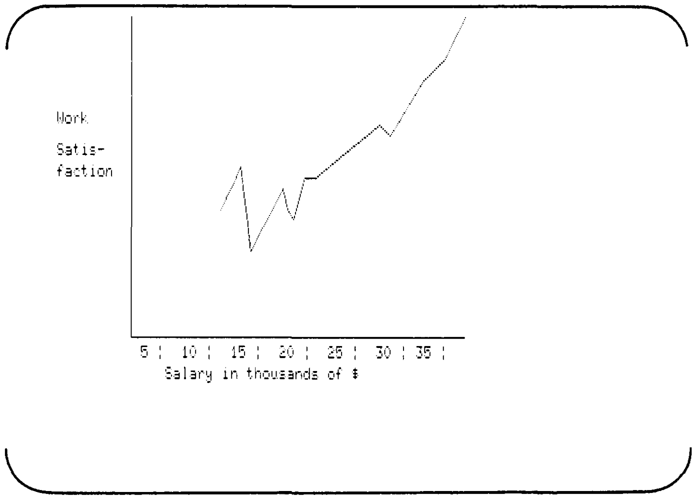

ZK-5524-86

Points are frequently less structured than those created by the circle program. Statistics from surveys, for example, could provide the data for a line chart. The following program plots the data gathered from an informal survey of supermarket employees and uses DATA and READ statements to fill the coordinate arrays. When data is maintained in a file, you can open the file in the usual manner within your graphics program. The employees' salaries (divided by 100,000) provide the data for the x-array. Employees rated their job satisfaction on a scale of 0 to 1; this provides the data for the y-array. The DATA statement provides the salar and work satisfaction rating for each of the fifteen employees surveyed. The chart illustrates the relationship between salary earned and employee job satisfaction.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG Counter
DIM SINGLE x_array(14),y_array(14)

!+
!Draw the axes
!
GRAPH LINES 0.225,1.0; 0.225,0.2; 1.0,0.2
!+
!Read in the data
!-
DATA 0.43,0.5, 0.475,0.6, 0.5,0.4, 0.55,0.5,         &
     0.575,0.55, 0.585,0.5, 0.6,0.475, 0.625,0.575,  &
     0.65,0.575, 0.8,0.7, 0.825,0.575, 0.9,0.8,      &
     0.925,0.825, 0.95,0.85, 1.0,0.95
READ x_array(Counter), y_array(Counter) FOR Counter = 0% TO 14%
!+
!Draw the lines
!-
MAT GRAPH LINES x_array, y_array
!+
!Add the text
!-
GRAPH TEXT AT 0.225,0.15 : " 5 |  10 |  15 |  20 |  25 |  30 | 35 |"
GRAPH TEXT AT 0.3,0.1 : "Salary in thousands of $"
GRAPH TEXT AT 0.05,0.7 : "Work"
GRAPH TEXT AT 0.05,0.625 : "Satis-"
GRAPH TEXT AT 0.05,0.575 : "faction"
END
```

Output



Chapter 4 illustrates how to change the text attributes such as the justification and text path so that graphs can be accurately labeled.

.8 Graphics Subprograms

Graphics images can be defined in separate blocks of code known as *pictures*. A picture can be compiled separately and must be delimited by a PICTURE and an END PICTURE statement, for example:

```
PICTURE unique_name
    .
    .
    .
END PICTURE
```

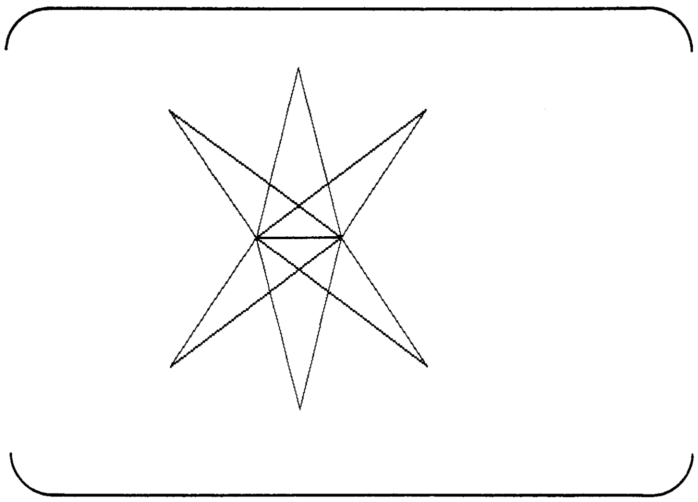
As with other subprograms, pictures should be declared with an EXTERNAL statement in the main program. To invoke a picture, you use the DRAW statement, not the CALL statement, for example:

```
PROGRAM display
  EXTERNAL PICTURE sample
  DRAW sample
END PROGRAM
```

You can pass parameters to pictures as with any other subprogram. The following example invokes the picture *triangle* several times with the DRAW statement. The coordinates of the third point for the triangle are passed as parameters with each invocation of the picture.

```
PROGRAM display
  EXTERNAL PICTURE triangle(SINGLE,SINGLE)

  DRAW triangle(0.2,0.8)
  DRAW triangle(0.5,0.9)
  DRAW triangle(0.8,0.8)
  DRAW triangle(0.8,0.2)
  DRAW triangle(0.5,0.1)
  DRAW triangle(0.2,0.2)
END PROGRAM

PICTURE triangle (SINGLE x,y)
 GRAPH LINES 0.4,0.5; 0.6,0.5; x,y; 0.4,0.5
END PICTURE
```

Output


ZK-5506-86

The DRAW statement can include several useful options to invoke pictures in a variety of ways. Chapter 6 discusses pictures in detail and illustrates how to create complex images with the DRAW statement. Simple pictures are used where appropriate in Chapters 3 through 5.

9 Summary

This chapter has explained how to use the following statements:

- GRAPH POINTS
- GRAPH LINES
- GRAPH AREA
- GRAPH TEXT
- PLOT LINES
- MAT GRAPH POINTS
- MAT GRAPH LINES
- MAT GRAPH AREA

This chapter has also introduced pictures and how to invoke them with the DRAW statement. These concepts are illustrated throughout the manual and expanded fully in Chapter 6.

The points, lines, areas, and text in this chapter all display the VAX BASIC default attributes. VAX BASIC provides many statements to change attributes from their default values. The following chapters show you how to enhance your graphics displays by changing these attributes.

Chapter 3

Changing the Appearance of Graphics Objects

You can control the exact appearance of points, lines, and areas by setting attribute values. This chapter shows you how to change the color, size, and style of these graphics objects. For each graphics object, this chapter does the following:

- Illustrates the available attribute values
- Demonstrates how to select the attributes
- Shows how to ask about the attribute values
- Provides a table of available attributes and their values

Text attributes are discussed separately in Chapter 4.

1 Graphics Attributes

The examples in Chapter 2 illustrate the default values for the attributes of each graphics object displayed. You can change these attributes with SET statements, and you can inquire what the current values are with ASK statements. The default attributes are displayed unless you have supplied alternative values with a SET statement. Table 3–1 shows the attributes you can select for each of the graphics objects. Note that SET statements can be used in immediate mode in the environment if SET is preceded with a backslash (\).

Table 3–1: Graphics Objects and Their Attributes

Graphics Object	Attributes
Point	Style, color, and size of the marker
Line	Style, color, and width of the line
Area	Interior fill styles, and color of the fill

All attributes are not necessarily supported by each hardware device. Fo the most part, this chapter shows examples of the available attributes foı VT125 and VT240 terminals.

3.2 Setting the Color

The colors red, green, and blue are available by default on your color monitor in addition to the background color. You can display these four colors on your screen at the same time. Some devices, such as the VAXstation II/GPX, are more versatile; the examples in this section show output displayed on a VT240 terminal.

Many factors affect the display and perception of foreground colors; the colors displayed in this manual may not be the same as those displayed on the device you use. The brightness and contrast controls change the perceived color, as do the available room light and any previously define terminal color setup. Note that if you have a monochrome monitor, the color statements may affect the shading of graphics objects displayed on your screen.

Each color has a numeric index value associated with it. Table 3–2 lists the default color associated with each color index for VT125 and VT240 terminals. Color 0 represents the background color, or the color of a display surface that is clear. Colors 1, 2, and 3 represent foreground colors. If you have a color terminal, make sure that you assign the corre device type number to GKS$WSTYPE, as described in Chapter 1. You may have to reset your terminal color setup after executing a graphics program.

Table 3–2: Default Colors for VT125 and VT240 Color Terminals

Index	Color
0	Black
1	Green
2	Red
3	Blue

You use these numeric values to change the colors in your displays.

?.1 Changing the Color Index

When you supply a numeric value to one of the SET . . . COLOR statements, the value is used to determine the color of any subsequent points, lines, filled areas, or text. For instance, each of the following statements sets the color of the objects specified to blue.

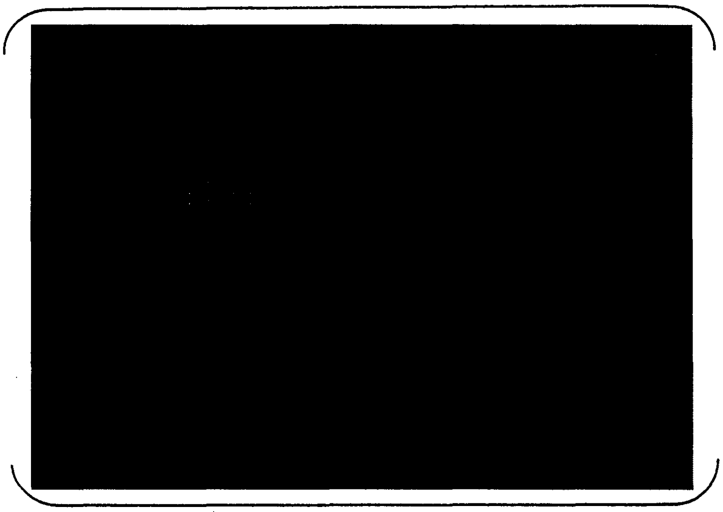
```
SET POINT COLOR 3

SET LINE COLOR 3

SET AREA COLOR 3

SET TEXT COLOR 3
```

You can set the color before executing a graphics output statement. The color you set continues to be the foreground color for that graphics object until you change it with another SET . . . COLOR statement. (Note that if you set the foreground color to the same color as the background, you will not be able to distinguish the display from the background.) The following example sets the color for the lines, area, and text in a simple diagram.

Example

```
!+
!Outline the book
!-
SET LINE COLOR 1
GRAPH LINES  0.26,0.8; 0.43,0.8; 0.43,0.25; 0.26,0.25; 0.26,0.8;  &
             0.35,0.88; 0.48,0.88; 0.48,0.33; 0.43,0.25           &
GRAPH LINES  0.43,0.8; 0.48,0.88
!+
!Fill spine
!-
SET AREA COLOR 3
GRAPH AREA: 0.43,0.8; 0.43,0.25; 0.48,0.33; 0.48,0.88

SET TEXT COLOR 2
GRAPH TEXT AT 0.265,0.65 : "Your"
GRAPH TEXT AT 0.26,0.6 : "BASIC"
GRAPH TEXT AT 0.265,0.55 : "book"
```

Output

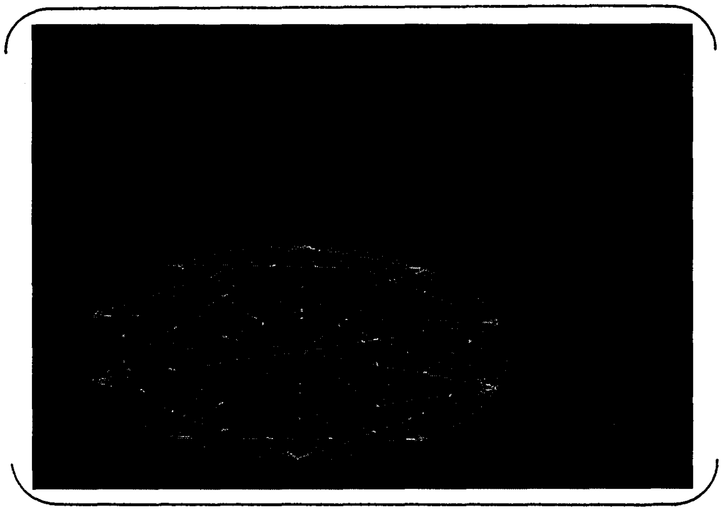


The design displayed by the following example is achieved with a SET LINE COLOR statement inside a loop to vary the color. The program connects points on the perimeter of an oval.

Example

```
OPTION TYPE = EXPLICIT
OPTION ANGLE = RADIANS
DECLARE LONG loop_count, inner,       &
             loop_index, counter,     &
             color_no,                &
        SINGLE turn
DIM SINGLE x_array(40),y_array(40),   &
           x2_array(1),y2_array(1)

turn = 0
!+
!Fill arrays with coordinates for points on perimeter
!-
FOR loop_count = 0 TO 40
    turn = 2 * PI * loop_count/40
    x_array(loop_count)  = (SIN(turn) + 1)/2
    y_array(loop_count)  = (COS(turn) + 1)/4
NEXT loop_count
!+
!Draw the perimeter
!-
MAT GRAPH LINES x_array, y_array
color_no = 1
!+
!Select points on perimeter
!-
FOR loop_index = 0 TO 40 STEP 4
    x2_array(counter) = x_array(loop_index)
    y2_array(counter) = y_array(loop_index)
    !+
    !Inner loop to connect other points to
    !the point x2_array(counter),y2_array(counter)
    !-
    counter = counter + 1
    FOR inner = 0 TO 40 STEP 4
        x2_array(counter) = x_array(inner)
        y2_array(counter) = y_array(inner)
        !+
        !Keep the color 1, 2, or 3
        !-
        IF color_no = 4
           THEN color_no = 1
           END IF
        SET LINE COLOR color_no
        color_no = color_no + 1
        !+
        !Draw all the lines connecting selected point to others
        !-
        MAT GRAPH LINES x2_array, y2_array
  NEXT inner
NEXT loop_index
END
```

Output



3.2.2 Changing the Color Intensities

The four index colors are each defined by the intensities of red, green, an blue. The default intensity values for the color indices are as follows:

Color Index	Color	Red Intensity	Green Intensity	Blue Intensity
0	Black	0.0	0.0	0.0
1	Green	0.0	1.0	0.0
2	Red	1.0	0.0	0.0
3	Blue	0.0	0.0	1.0

You can change the actual color displayed for each index by changing the red, green, and blue intensities with the SET COLOR MIX statement. When you use this statement, you alter the default color of the index displayed on your screen. For instance, although the color index 3 displa blue by default, you can change the color that this index displays by changing the intensity values for red, green, and blue while index 3 is selected. By supplying alternative values for the intensities, you can greatly increase your choice of possible foreground colors. (The device you use may not have the capability to display more than four of these at the same time.) The values for these intensities can vary from 0 through 1.

The following statement changes the color index 3 to be raspberry instead of the default blue.

Example

```
DECLARE SINGLE red_var, green_var, blue_var
red_var = 0.84
green_var = 0.0
blue_var = 0.56
SET COLOR MIX , INDEX 3 : red_var, green_var, blue_var
```

Remember that color index values are device dependent. Values for VT125 and VT240 terminals and for VAXstations can be found in Appendix B in this manual. Values for other devices can be found in the VAX GKS documentation and in the documentation for the particular device you use.

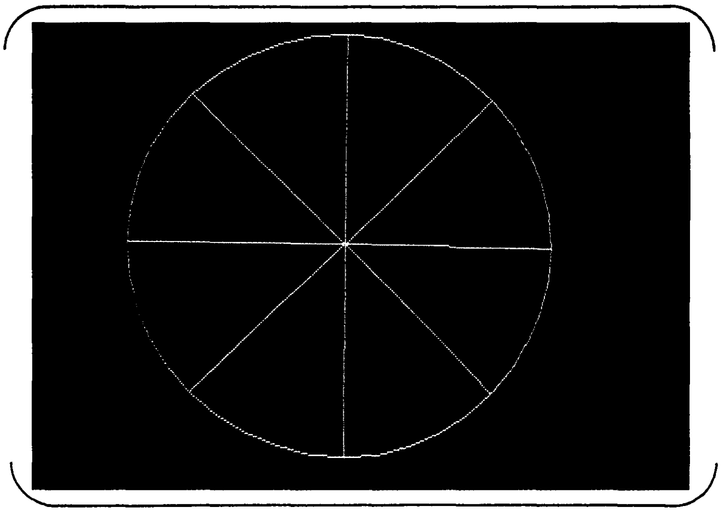
The following program illustrates the use of the SET COLOR MIX statement. Each time the graphics object (a wheel) is drawn, the color index is set to 2; however, the default color of red is changed. Instead, the values of the color intensities are changed for each display of the wheel, and in each case the diagram is presented with a changed color. Only one screen is illustrated after the example.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE Wheel
DECLARE SINGLE green, blue, red
!+
!Set up the variations in color intensities
!-
SET LINE COLOR 2
red = 0.0
green = 0.0
blue = 0.56
!+
!Set index 2 to a violet shade
!-
SET COLOR MIX , INDEX 2 : red, green, blue
DRAW Wheel
SLEEP 5%
CLEAR
```

```
!+
!Set up index 2 as orange
!-
red = 0.8538
green = 0.6646
blue = 0.2862
SET COLOR MIX , INDEX 2 : red, green, blue
DRAW Wheel
END

PICTURE Wheel
  DECLARE SINGLE loop_count, loop_index,        &
                 turn,x_center, y_center,x,y
  DIM SINGLE x_array(40),y_array(40)
  !+
  !Fill the arrays and draw the circle
  !-
  turn = 0
  FOR loop_count = 0 TO 40
      turn = 2 * PI * loop_count/40
      x_array(loop_count) = (SIN(turn) + 1)/2
      y_array(loop_count) = (COS(turn) + 1)/2
  NEXT loop_count
  MAT GRAPH LINES  x_array, y_array
  Draw_spokes:
  !+
  !Select 8 points on circle for spokes
  !-
  FOR loop_index = 5 TO 40 STEP 5
      x = x_array(loop_index)
      y = y_array(loop_index)
      x_center = 0.5
      y_center = 0.5
      GRAPH LINES x,y ; x_center,y_center
  NEXT loop_index
END PICTURE
```

Output



The following example illustrates many of the possible shades that you can display on your terminal. The wheel is drawn within a set of three loops that change the intensities of red, green, and blue for the current color index. Some of the colors are close to the background color; therefore, if you run this program, you should turn up the brightness on your terminal. (Be aware that this program runs for several minutes.)

Example

```
DECLARE SINGLE green, blue, red
EXTERNAL PICTURE wheel
color_var = 1
!+
!Set up the variations in color intensities
!-
FOR green = 0.2 TO 1 STEP 0.2
   FOR red = 0.2 TO 1 STEP 0.2
      FOR blue = 0.2 TO 1 STEP 0.2
         SET COLOR MIX , INDEX color_var : red, green, blue
         SET LINE COLOR color_var
         !+
         !For every iteration of this loop draw the wheel
         !-
         DRAW wheel
         SLEEP 2%
         CLEAR
      NEXT blue
   NEXT red
NEXT green
END
```

Remember that if you do not change the values for the intensities, the index values display the default index colors.

3.2.3 Asking About Color Attributes

ASK statements allow you to retrieve the current value of attributes. VAX BASIC provides two ASK statements that allow you to retrieve the information about the color attributes.

The ASK MAX COLOR statement allows you to retrieve the highest valid color index value for the particular device you use. In the following statement, the highest color index number available for that device is assigned to the variable *highest_poss.*

```
ASK MAX COLOR highest_poss
```

For instance, if the value returned is 10, this means that the highest index value you can set is 10. This does not necessarily mean that there are 10 different indices on the device you are using because the color indices need not be contiguous. (VAX BASIC signals a warning when you open a device that does not have contiguous color indices.) Consult the documentation for your particular device for details.

The ASK COLOR MIX statement retrieves the current values for the intensities of red, green, and blue, in that order, for the particular color index that you specify. The following statement asks for the current intensities for red, green, and blue when the color index is set to 2. The values for the intensities are placed in the three variables you supply. The assigned values are the defaults unless a SET COLOR MIX statement has been executed.

```
ASK COLOR MIX  , INDEX 2 : red_var, green_var, blue_var
```

.3 Setting Point Attributes

In addition to changing the color of points, you can change the style and the size. When you display a point with the default attributes, the point is an asterisk (*) of the smallest possible size your device can display.

.3.1 Changing the Point Style

So far the examples have displayed an asterisk to mark a point on the screen. As shown in Table 3–3, VAX BASIC provides five point markers, each with a numeric value. At the start of program execution, the point style value is 3. The number of available marker styles varies according to the device you use.

Table 3–3: Point Styles

Value	Point Style
1	Dot
2	Plus sign
3	Asterisk (default)
4	Circle
5	Diagonal cross

To display a point style other than the default, you use the SET POINT STYLE statement. You must supply a numeric value for the point style you want. For example, the following statements set the point style to be a plus sign and a dot respectively:

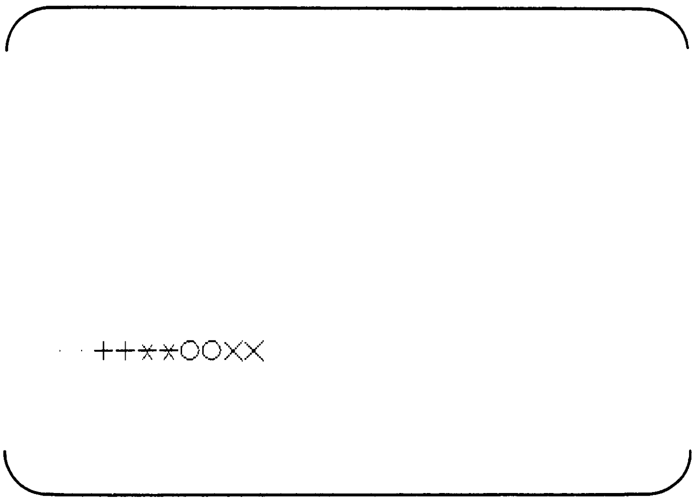
```
SET POINT STYLE 2
SET POINT STYLE 1
```

To change the point style, the SET POINT STYLE statement must preced the output statement that displays the points. The point style you set remains in effect until VAX BASIC executes another SET POINT STYLE statement. To regain the default point style, you must set the point style back to the value 3 for the asterisk.

Example

```
PROGRAM point_styles
  !+
  !Set the style to a dot
  !-
  SET POINT STYLE 1
  GRAPH POINTS 0.1,0.2; 0.15,0.2
  !+
  !Set the style to a plus sign
  !-
  SET POINT STYLE 2
  GRAPH POINTS 0.2,0.2; 0.25,0.2
  !+
  !Set the style to an asterisk
  !-
  SET POINT STYLE 3
  GRAPH POINTS 0.3,0.2; 0.35,0.2
  !+
  !Set the style to a circle
  !-
  SET POINT STYLE 4
  GRAPH POINTS 0.4,0.2; 0.45,0.2
  !+
  !Set the style to a diagonal cross
  !-
  SET POINT STYLE 5
  GRAPH POINTS 0.5,0.2; 0.55,0.2
END PROGRAM
```

Output


ZK-4904-86

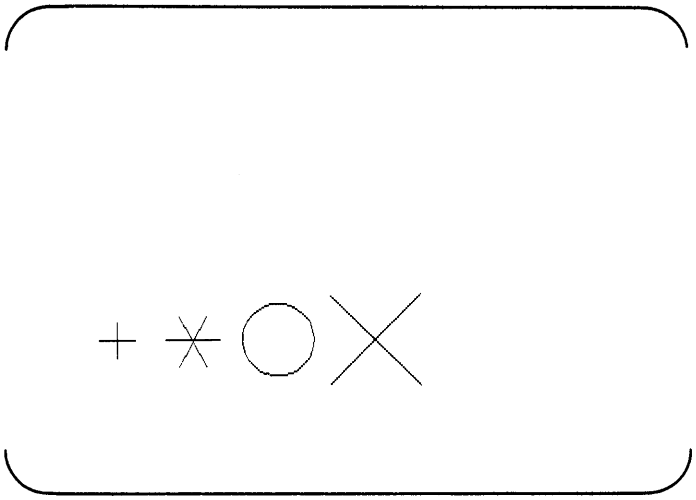
.3.2 Changing the Point Size

You can increase the size of each point style except the dot. To do this, supply the SET POINT SIZE statement with a numeric value for the scale of the marker. Possible point size values depend on the device. For example, valid values for VT125 and VT240 terminals are 1 (the default) through 12. The new point size remains in effect until another SET POINT SIZE statement is executed. Note that the dot is always displayed in the smallest possible size, size 1. The following example shows a few of the point sizes available on VT240 terminals.

Example

```
DECLARE LONG loop_count
FOR loop_count = 1 TO 5
   SET POINT STYLE loop_count
   SET POINT SIZE loop_count
   GRAPH POINTS loop_count/5,0.3
NEXT loop_count
END
```

Output



ZK-4896-86

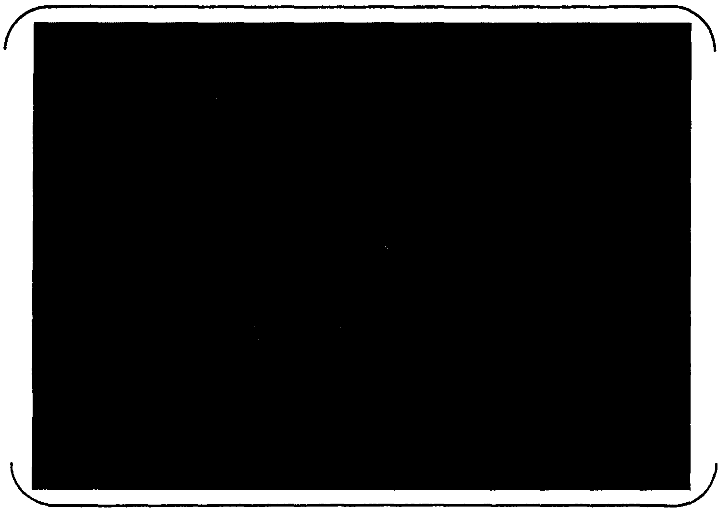
The following example marks each of the vertices of a hexagon with a size 1 diagonal cross and then marks the same points with size 7 circle markers.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE x,y,                  &
        LONG loop
!+
!Supply data for points of hexagon
!-
DATA 0.7,0.5, 0.6,0.3, 0.4,0.3,      &
     0.3,0.5, 0.4,0.7, 0.6,0.7
```

```
!+
!Procedure for each point
!-
FOR loop = 0 TO 5
    READ x,y
    !+
    !Draw a size 1 diagonal cross at x,y
    !-
    SET POINT STYLE 5
    SET POINT SIZE 1
    SET POINT COLOR 2
    GRAPH POINTS x,y
    !+
    !Draw a size 7 circle marker at x,y
    !-
    SET POINT STYLE 4
    SET POINT SIZE 7
    SET POINT COLOR 3
    GRAPH POINTS x,y
NEXT loop
END
```

Output



3.3.3 Asking About Point Attributes

You can retrieve information about the point style and color by supplying integer variables with the ASK POINT statements. The following statements retrieve the values for the point style and color respectively.

```
ASK POINT STYLE style_var
ASK POINT COLOR color_var
```

The values are returned in the variables you supply. For instance, after th ASK POINT STYLE statement in the following example is executed, the current style value is stored in the variable *style_var*. The value returned is the value used if an appropriate output statement were to be executed a this time. This feature allows you to store attribute values, change them, and reset them to the stored values at another point in your program.

Example

```
DECLARE LONG style_var, color_var
EXTERNAL PICTURE Multi_points(LONG,LONG)
GRAPH POINTS 0.1,0.85; 0.2,0.85; 0.3,0.85; 0.4,0.85
!+
!Ask for the value of the point style and color
!-
ASK POINT STYLE style_var
ASK POINT COLOR color_var
SET POINT STYLE 4
SET POINT COLOR 3

!+
!Invoke a picture using the new values
!-
DRAW Multi_points(style_var, color_var)
!+
!Reset the attributes to the stored values
!-
SET POINT STYLE style_var
SET POINT COLOR color_var
    .
    .
    .
```

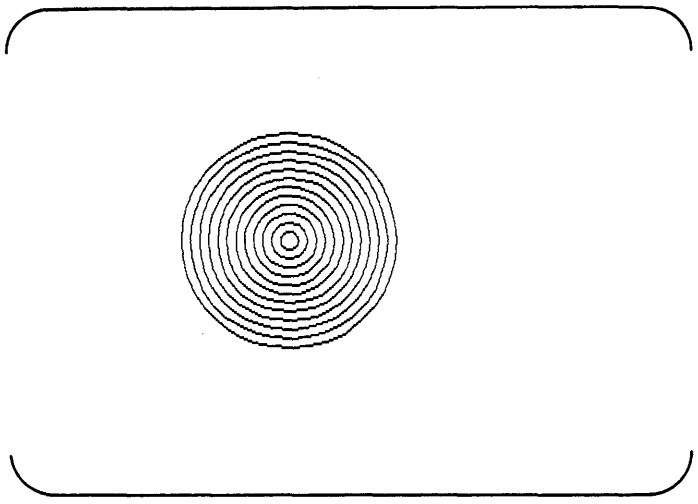
For more information about pictures, see Chapter 6.

The ASK MAX POINT SIZE statement allows you to retrieve the largest possible value for the point size for your particular device. This value is stored in a variable you supply. The example shows the largest value used in a loop to draw concentric circles using point style 4.

Example

```
DECLARE LONG maximum,loop_counter
ASK MAX POINT SIZE maximum
SET POINT STYLE 4
FOR loop_counter = 1 TO maximum
    SET POINT SIZE loop_counter
    GRAPH POINTS 0.5,0.5
NEXT loop_counter
END
```

Output



ZK-5507-86

3.4 Setting Line Attributes

As with points, you can change the style, size, and color of lines. The effects of these changes vary depending on the type of device you use.

3.4.1 Changing the Line Style

VAX BASIC provides the four different line styles described in Table 3–4. At the start of program execution, the line style is value 1, solid.

Table 3–4: Line Styles

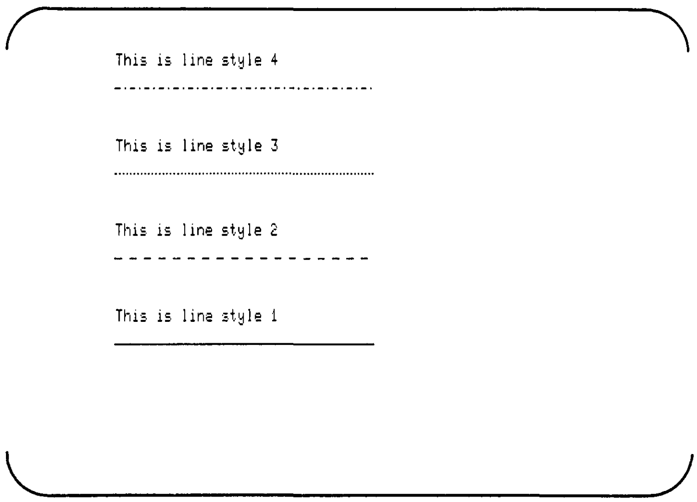
Value	Line Style
1	Solid (default)
2	Dashed
3	Dotted
4	Dashed-dotted

To change the line style, supply the selected value to the SET LINE STYLE statement. This example displays each of the four line styles.

Example

```
DECLARE LONG loop_count,              &
        SINGLE y
FOR loop_count = 1 TO 4
    SET LINE STYLE loop_count
    y = loop_count/10
    GRAPH TEXT AT 0.1,(y + 0.05) : "This is line style " + STR$(loop_count)
    GRAPH LINES 0.1,y; 0.45,y
NEXT loop_count
END
```

Output



ZK-4905-86

.4.2 Changing the Line Width

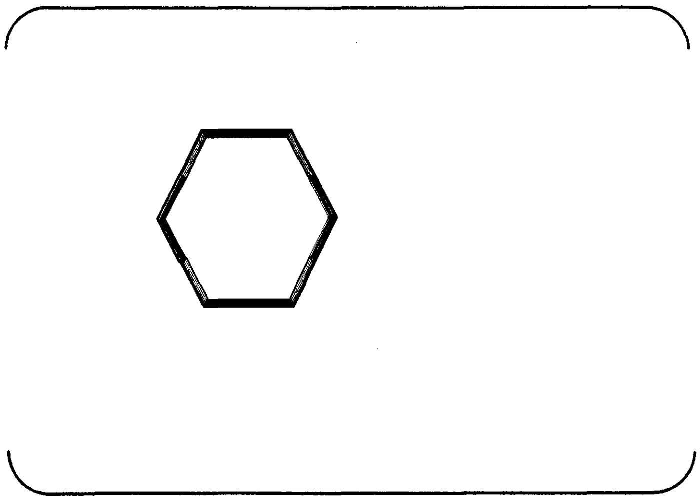
You can increase the width of lines just as you can increase the size of points. To do this, use the SET LINE SIZE statement.

The output of the following example shows a solid line increased to sizes 10, 20, and 60 respectively. When the line width is increased to a large size, the outer edges of closed figures no longer meet, as is the case when the hexagon lines are increased to size 60. Open figures can be increased without this apparent distortion. The amount of distortion that large sizes produce depends on the capabilities of your terminal.

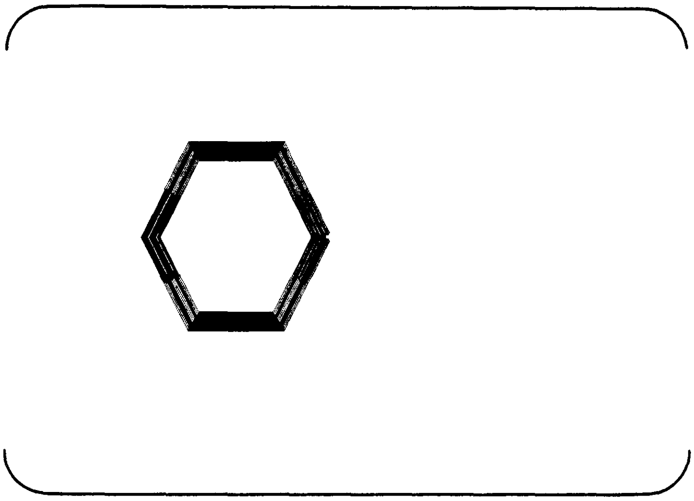
Example

```
EXTERNAL PICTURE Hexagon(LONG)
DRAW Hexagon(10)
SLEEP 5%
CLEAR
DRAW Hexagon(20)
SLEEP 5%
CLEAR
DRAW Hexagon(60)
END

PICTURE Hexagon(LONG line_size)
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE x_array(6),y_array(6)
  DECLARE LONG loop
  DATA 0.7,0.5, 0.6,0.3, 0.4,0.3,            &
       0.3,0.5, 0.4,0.7, 0.6,0.7, 0.7,0.5
  READ x_array(loop),y_array(loop) FOR loop = 0 TO 6
  SET LINE SIZE line_size
  MAT GRAPH LINES x_array, y_array
END PICTURE
```

Output Screen 1:

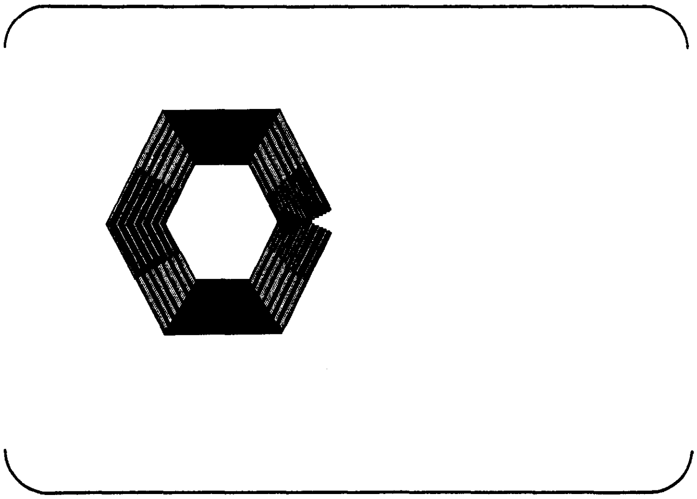


Output Screen 2:



ZK-4898-86

Output Screen 3:



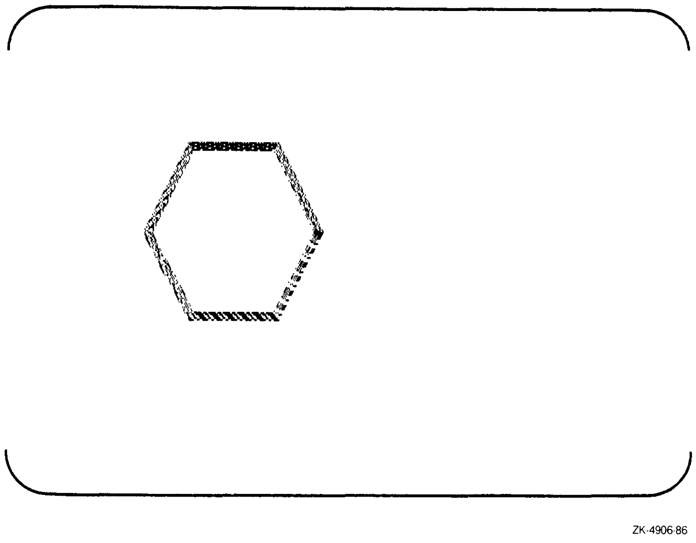
ZK-4897-86

By changing both the line size and the line style, you can achieve some interesting results. Here, the line size is 10 and the line style is value 4, dashed-dotted.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE Hexagon(LONG)
SET LINE STYLE 4
DRAW Hexagon(10)
END
```

Output



3.4.3 Asking About Line Attributes

You can retrieve the value of the current line style with the ASK LINE STYLE statement. Here, GRAPH LINES draws a line with the style value stored previously in the variable *style_value*.

Example

```
   .
   .
   .
ASK LINE STYLE style_value
SET LINE STYLE 3
   .
   .
   .
SET LINE STYLE style_value
GRAPH LINES 0.5,0.8; 0.2,0.4; 0.3,0.3
```

You can also retrieve the current line color with the ASK LINE COLOR statement.

Example

```
DECLARE LONG which_color
ASK LINE COLOR which_color
IF which_color = 1
   THEN
       SET LINE COLOR 3
   ELSE
       SET LINE COLOR 2
END IF
   .
   .
   .
```

The largest possible line size for your device can be retrieved with the ASK MAX LINE SIZE statement. You must supply a numeric variable.

```
ASK MAX LINE SIZE thickest
```

5 Setting Area Attributes

Areas can be represented by four different interior fill styles. Two of the interior fill styles, pattern and hatch, can produce a wide variety of representations of an area. To make these changes, you use the SET AREA STYLE and the SET AREA STYLE INDEX statements.

3.5.1 Changing the Fill Style

VAX BASIC provides four different interior fill styles for areas:

- SOLID (the default)
- HOLLOW
- PATTERN
- HATCH

At the start of program execution, the interior fill style is solid.

Unlike other attributes, the interior fill styles do not have numerical values. Instead, you supply the SET AREA STYLE statement with a strir expression equivalent to one of the interior fill style names. You can enter strings in either upper- or lowercase. Both of the SET AREA STYL statements in the following example set the interior fill style to hollow.

Example

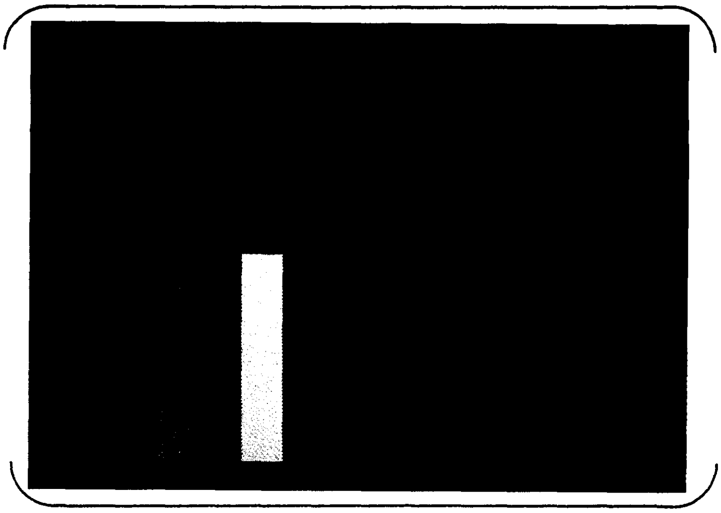
```
DECLARE STRING CONSTANT hollow_fill = "HOLLOW"
    .
    .
    .
SET AREA STYLE hollow_fill
MAT GRAPH AREA x_array, y_array
SET AREA STYLE "hollow"
MAT GRAPH AREA x_array, y_array
```

The following example displays each of the four interior fill styles. As the pattern style mixes only colors, the difference between the solid and pattern styles may not be discernible on a monochrome display.

Example

```
SET AREA STYLE "Hollow"
GRAPH AREA 0.0,0.5; 0.1,0.5; 0.1,0.0; 0.0,0.0

SET AREA STYLE "Solid"
GRAPH AREA 0.2,0.5; 0.3,0.5; 0.3,0.0; 0.2,0.0

SET AREA STYLE "Pattern"
GRAPH AREA 0.4,0.5; 0.5,0.5; 0.5,0.0; 0.4,0.0
SET AREA STYLE "Hatch"
GRAPH AREA 0.6,0.5; 0.7,0.5; 0.7,0.0; 0.6,0.0
END
```

Output



.5.2 Changing Pattern and Hatch Styles

The interior fill styles for pattern and hatch can be changed with the SET AREA STYLE INDEX statement. The pattern or hatch style displayed depends on the setting of the area style index. Table 3–5 lists the various styles you can display by changing the index.

Table 3–5: VT125 and VT240 Pattern and Hatch Style Inde Values

Fill Style	Index Value	Display
HATCH	1 (default)	Cross hatched
	2	Hatching with lines at 45 degrees
	3	Hatching with lines at -45 degrees
	4	Hatching with horizontal lines
	5	Hatching with vertical lines
	6	Hatching with lines at 45 degrees (sparse)
	7	Hatching with lines at -45 degrees (sparse)
	8	Hatching with horizontal lines (sparse)
	9	Hatching with vertical lines (sparse)
	10–19	Varying density hatches[1]
	33–126	Hatching with ASCII character[2]
PATTERN[3]	1 (default)	Mixes colors 1 and 2
	2	Mixes colors 2 and 3
	3	Mixes colors 3 and 1
	4	Mixes colors 0 and 1
	5	Mixes colors 0 and 2
	6	Mixes colors 0 and 3

[1]A varying density hatch style is a mixture of pixels of a given color and white pixels, which give the appearance of different shades. For hatch styles 10 through 19, the hatches progress from light to darker; 10 being the lightest and 19 being the darkest.

[2]The style index values 33 through 126 generate hatching with the corresponding character from th ASCII table. For example, the ASCII value for the character A is 65; a hatched fill area with style index 65 fills the area with a number of uppercase As.

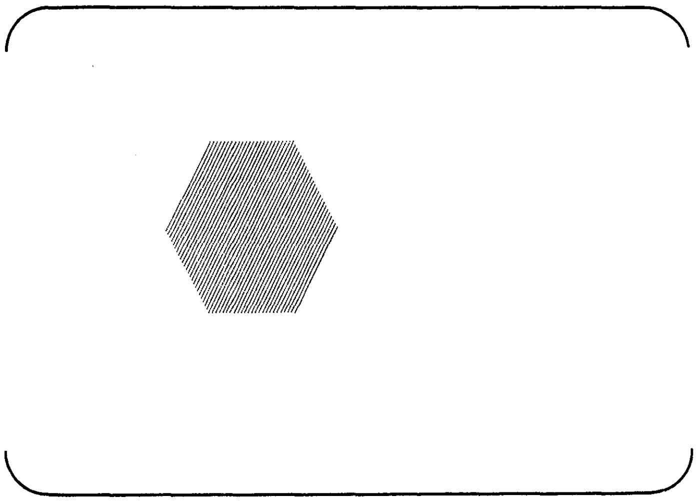
[3]Pattern style index values are appropriate only for color monitors.

The default style index value is 1 for both hatch and pattern. Samples of the default pattern and hatch fill styles are shown in Section 3.5.1. The available fill styles for pattern and hatch vary according to the device you use; Appendix B lists the styles available for VAXstations. The following example displays three of the many hatch styles available on VT125 and VT240 terminals.

Example

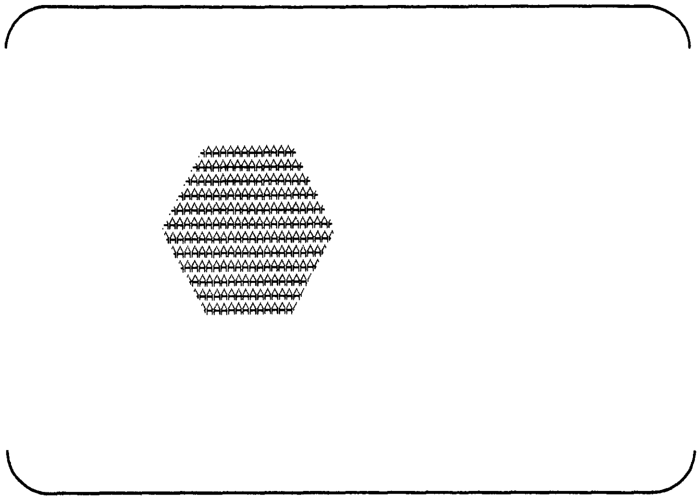
```
EXTERNAL PICTURE Draw_hatch(LONG)
SET AREA STYLE "HATCH"
DRAW Draw_hatch(3)
SLEEP 3%
CLEAR
DRAW Draw_hatch(65)
SLEEP 3%
CLEAR
DRAW Draw_hatch(72)
END

PICTURE Draw_hatch(LONG style_index)
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE x_array(5),y_array(5)
  DECLARE LONG loop
  DATA 0.7,0.5, 0.6,0.3, 0.4,0.3,            &
       0.3,0.5, 0.4,0.7, 0.6,0.7
  READ x_array(loop),y_array(loop) FOR loop = 0 TO 5
  SET AREA STYLE INDEX style_index
  MAT GRAPH AREA x_array, y_array
END PICTURE
```

Output Screen 1:

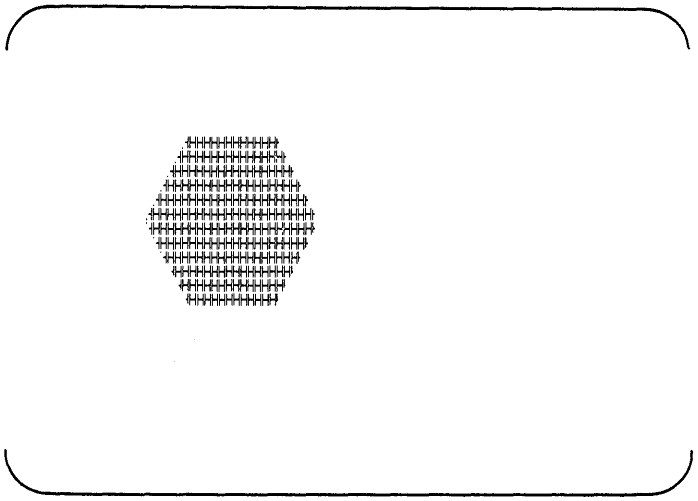


ZK-4902-86

Output Screen 2:


ZK-4901-86

Output Screen 3:


ZK-4900-86

5.3 Asking About Area Attributes

The ASK AREA STYLE statement lets you retrieve the value of the current style. This value is placed in a string variable you supply.

Example

```
DECLARE STRING which_style
    .
    .
    .
ASK AREA STYLE which_style
GRAPH TEXT AT 0.1,0.5 : "The interior fill style is " + which_style
```

Similarly, the ASK AREA STYLE INDEX statement retrieves the current index value of the pattern or hatch style. You can use this value to reset the index after branching to another program module, as shown in the following example. The example also stores the initial area color and fill style. You must supply variables of the appropriate data type with each of these statements.

Example

```
PROGRAM Welcome
  EXTERNAL PICTURE Menu_setup
  DECLARE LONG index_number, which_color
  DECLARE STRING fill
  ASK AREA STYLE fill
  ASK AREA STYLE INDEX index_number
  ASK AREA COLOR which_color
  !+
  !Branch to another module
  !-
  DRAW Menu_setup
      .
    .
    .
  !+
  !Return to main program, reset the area attributes
  !-
  SET AREA STYLE fill
  SET AREA STYLE INDEX index_number
  SET AREA COLOR which_color
    .
    .
    .
END PROGRAM
```

3.6 Summary

This chapter has explained how to use the following statements:

- SET/ASK . . . COLOR
- SET/ASK COLOR MIX
- SET/ASK . . . STYLE
- SET . . . SIZE and ASK MAX . . . SIZE

These statements allow you to set and inquire about the color, size, and style attributes of points, lines, and areas. The following chapter shows you how to change the attributes associated with graphics text.

Chapter 4

Changing Text Attributes

You can change the direction, height, width, spacing, angle, and justification of characters in different font designs. This chapter illustrates the many text attributes that you can access with VAX BASIC statements.

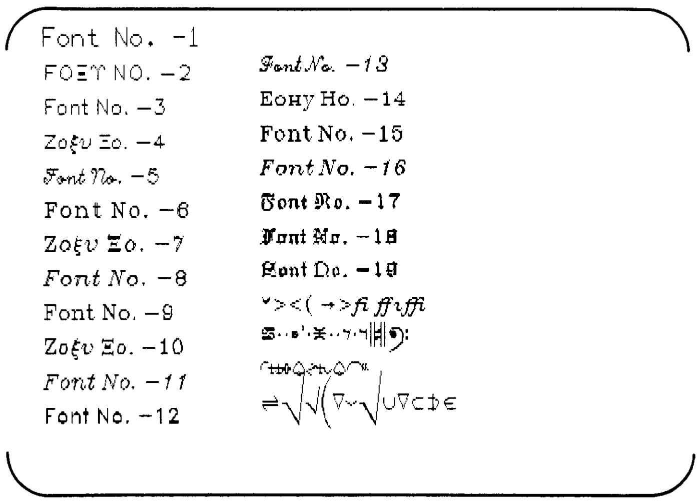
.1 Selecting a Font

VAX BASIC supports both hardware and software fonts. The number of available hardware fonts depends on the device you use. VAXstations support several hardware fonts, while VT125 and VT240 terminals support one hardware font each. If no hardware fonts are available for a device, software fonts are used by default. For illustrations of the VAXstation hardware fonts, see Appendix B. In general, hardware fonts are displayed faster than software fonts.

Software fonts provide a graphical representation of defined characters. A font can be defined as a complete set of monospaced or proportionally spaced characters of a particular printing type, size, and face.

The fonts displayed in this chapter are the fonts that are available on VT125 and VT240 terminals. These fonts are proportionally spaced and are known as the Hershey fonts, digitized by A.V. Hershey of the Naval Surface Weapons Laboratory. The default font and the Hershey fonts are illustrated in Figure 4–1. The values representing the fonts are -1 through -23, as illustrated. The default font is software font number -1.

Figure 4–1: VAX BASIC Software Fonts



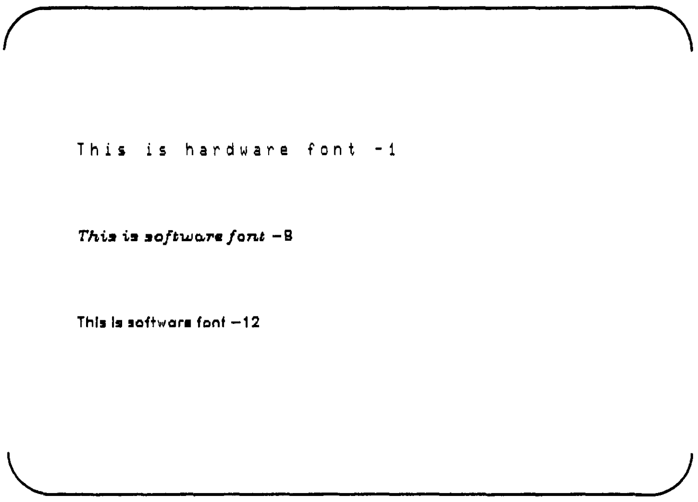
Note that none of the statements discussed in this chapter has any effect on output from the VAX BASIC statements PRINT and PRINT USING. In graphics programs, it is recommended that you use GRAPH TEXT statements, not PRINT statements.

You can access the different fonts easily with the SET TEXT FONT statement. This statement requires a number for the font you want to us In addition, you can optionally specify a string expression representing the degree of precision with which you want the characters drawn. The following example displays sample characters from the hardware font and two of the software fonts available on VT125 and VT240 terminals. Depending on the device you use, the hardware font may be displayed faster.

Example

```
SET TEXT FONT -1, "CHAR"
GRAPH TEXT AT 0.0,0.7 : "This is hardware font -1"
SET TEXT FONT -8 , "STROKE"
GRAPH TEXT AT 0.1,0.5 : "This is software font -8"
SET TEXT FONT -12 , "STROKE"
GRAPH TEXT AT 0.1,0.3 : "This is software font -12"
```

Output



ZK-5510-86

There are three possible values for the precision string:

- STRING
- CHAR
- STROKE (the default)

String precision provides the least degree of accuracy, while stroke provides the most.

Software fonts can be drawn only with stroke precision. If you request a software font and do not specify a precision string, VAX BASIC uses stroke precision by default. Stroke precision displays text characters as accurately as possible according to the specifications in the font design. Such features as the text height and the way text is *clipped* are affected by the precision. Attributes such as height, spacing, and the expansion factor are accurate for stroke precision. Software fonts are always drawn accurately regardless of the device; the accuracy with which hardware fonts are drawn is devic dependent.

You can supply one of two possible values for the precision string when you specify a hardware font: STRING and CHAR. Hardware fonts cannc be drawn with stroke precision; if you specify stroke precision for a hardware font, VAX BASIC displays the software font with the number that is closest to the font you specified.

The most significant trade-off between hardware and software fonts is speed; hardware fonts can be much faster than precise software fonts. If you want a faster display of characters, you should specify CHAR precision with the following statement:

```
SET TEXT FONT 1, "CHAR"
```

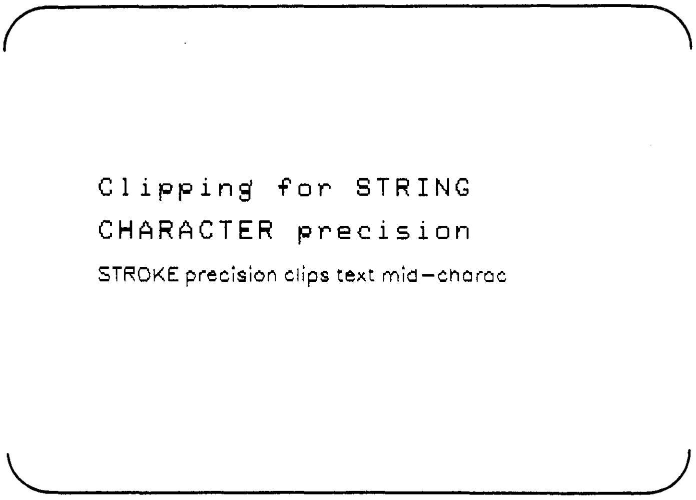
Character precision clips the text after the last complete character, while stroke precision clips the text exactly at the edge of the boundary of the display area, mid-character if necessary. With string precision, text is clipped either by character or by stroke precision depending on the capabilities of the device you use. The effect of each of these precision values may not be discernible in all cases, as the precision of text clippin; depends on the capabilities of the device you use. Clipping is discussed fully in Chapter 5.

The following example illustrates the effect of the precision value on the clipping of the text:

Example

```
SET TEXT HEIGHT 0.05
!+
!Select the hardware font for STRING precision
!-
SET TEXT FONT -1 , "STRING"
GRAPH TEXT AT 0.05,0.6 :                                          &
    "Clipping for STRING precision varies with each device"
!+
!Select the hardware font with CHAR precision
!_
SET TEXT FONT -1 , "CHAR"
GRAPH TEXT AT 0.05,0.5 :                                          &
    "CHARACTER precision clips text after complete characters"
!+
!Select software font -3 with STROKE precision
!-
SET TEXT FONT -3 , "STROKE"
GRAPH TEXT AT 0.05,0.4 :                                          &
    "STROKE precision clips text mid-character if necessary"
```

Output



ZK-5509-86

Note that if you request a software font with character or string precision, VAX BASIC displays the text with the hardware font that is closest in number to the font you requested. Similarly, if you request a software font that the device does not support, VAX BASIC displays the text with the hardware font that is closest in number to the font you requested.

You can retrieve the current value of the text font with the ASK TEXT FONT statement. In addition, you can optionally retrieve the current text precision value. The following statement retrieves the font number and the precision string in the variables *font_number* and *what_precision* respectively.

Example

```
DECLARE LONG which_font,         &
        STRING what_precision
ASK TEXT FONT which_font, what_precision
```

4.2 Setting the Character Height

The SET TEXT HEIGHT statement lets you set the height with which characters of text are drawn. The measurement for text height reflects the height of the uppercase letters compared to the height of the default drawing board: the height of the default drawing board is 1 unit, and th default height of uppercase characters is 0.035 on the same scale. When you change the height of the text characters, any subsequent GRAPH TEXT statement writes the text in the new height. Lowercase characters are drawn in proportion to the height of the uppercase characters according to the specifications in the font design. The following statement increases the height of the characters to a height of 0.05.

```
SET TEXT HEIGHT 0.05
```

The accuracy of the height in the display is affected by the precision use in a previous SET TEXT FONT statement. To achieve the exact height yo specify, text should be drawn with software fonts with STROKE precisio Hardware fonts with CHARACTER and STRING precision both map the height you request to the closest character size your device supports.

The ASK TEXT HEIGHT statement retrieves the current value for the tex height and stores the value in a real variable you supply. This value can be used to reset the height of characters at another point in your progran

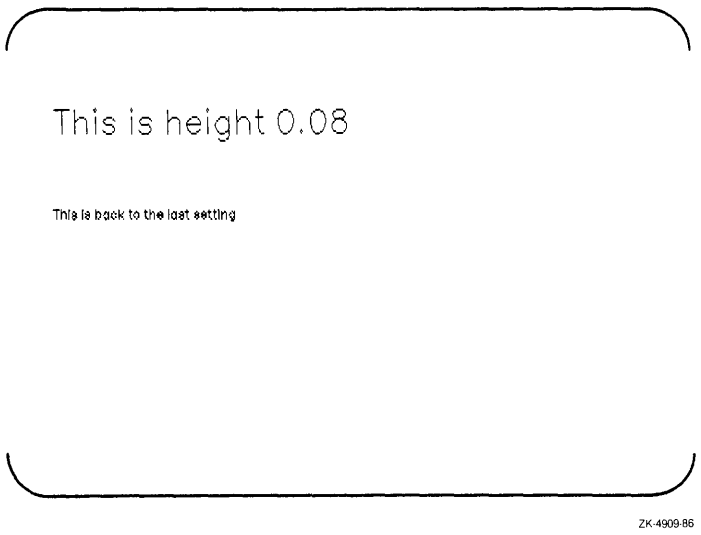
```
ASK TEXT HEIGHT height_var
```

The following example illustrates the use of the SET TEXT HEIGHT and ASK TEXT HEIGHT statements.

Example

```
DECLARE SINGLE old_height
ASK TEXT HEIGHT old_height
SET TEXT FONT -3, "STROKE"
SET TEXT HEIGHT 0.08
GRAPH TEXT AT 0.05,0.7 : "This is height 0.08"

SET TEXT HEIGHT old_height
GRAPH TEXT AT 0.05,0.5 : "This is back to the last setting"
```

Output



```
This is height 0.08

This is back to the last setting
```

When you change the window height with the SET WINDOW statement, you may also have to change the text height. For more information about changing the window height, see Chapter 5.

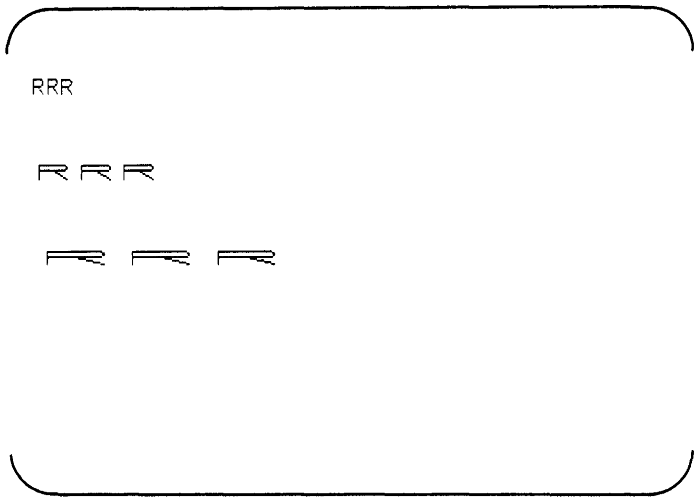
4.3 Setting the Character Height-to-Width Ratio

Each font has a specified width for each character directly related to the height. Therefore, when you change the height of a character, you automatically change the width proportionately. You can alter the normal proportions of each character with the SET TEXT EXPAND statement. Th initial value of the text width is 1.0, which displays characters with the height-to-width ratio specified in the font design. The following example demonstrates the use of the SET TEXT EXPAND statement by drawing th uppercase R and changing the height-to-width ratio. The ratio you suppl must be greater than zero.

Example

```
DECLARE STRING CONSTANT text_string = "RRR"
!+
!Set height but display normal proportions
!-
SET TEXT FONT -3 , "STROKE"
SET TEXT HEIGHT 0.05
GRAPH TEXT AT 0,0.8: text_string
!+
!Change the height-to-width value
!-
SET TEXT EXPAND 3
GRAPH TEXT AT 0,0.6 : text_string
!+
!Alter the height-to-width ratio, same height
!
SET TEXT EXPAND 6
GRAPH TEXT AT 0,0.4 : text_string
END
```

Output



ZK-4913-86

The ASK TEXT EXPAND statement lets you retrieve the current height-to-width ratio. You can use this value to reset the ratio later in your program.

Example

```
DECLARE SINGLE how_wide
ASK TEXT EXPAND how_wide
    .
    .
    .
SET TEXT EXPAND 2.0
    .
    .
    .
SET TEXT EXPAND how_wide
    .
    .
    .
```

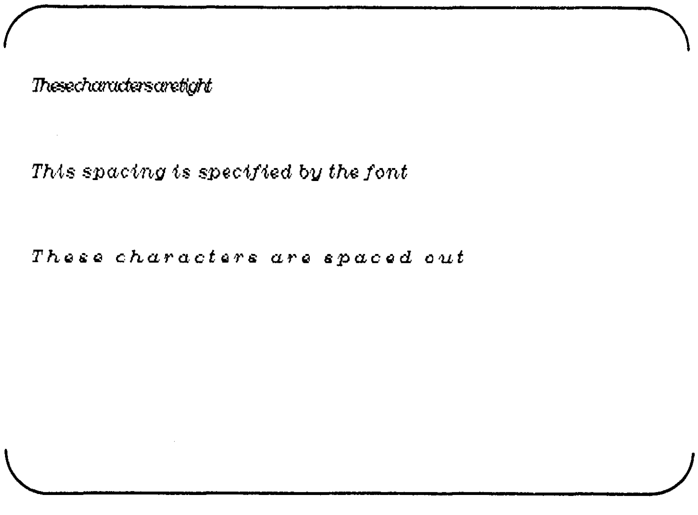
.4 Setting the Spacing Between Characters

Just as each font has a specified height-to-width ratio, each font has a specified amount of space between adjacent characters. The default space value between characters is zero and is set by the font design. You can change this value with the SET TEXT SPACE statement. Characters are drawn further apart when you supply this statement with a positive valu If you supply a negative value, the characters are drawn closer together. The following example illustrates the effect of the SET TEXT SPACE statement.

Example

```
SET TEXT FONT -8 , "STROKE"
SET TEXT HEIGHT 0.04

SET TEXT SPACE -0.2
GRAPH TEXT AT 0.1,0.8 : "These characters are tight"

SET TEXT SPACE 0
GRAPH TEXT AT 0.1,0.6 : "This spacing is specified by the font"

SET TEXT SPACE 0.2
GRAPH TEXT AT 0.1,0.4 : "These characters are spaced out"
```

Output



Thesecharactersaretight
This spacing is specified by the font
These characters are spaced out

ZK-4911-86

You can retrieve the current value for the spacing between characters wi the ASK TEXT SPACE statement, which assigns the value to the numeri variable you supply.

```
ASK TEXT SPACE spaced_out
```

.5 Setting the Text Angle

Text is normally drawn with a horizontal orientation; that is, the first character is placed on the left and subsequent characters are drawn to the right along a horizontal base. This is the default angle value of zero. Like other text attributes, the text angle can be changed. To do this, you supply the SET TEXT ANGLE statement with a numerical value for the text angle. The value can be in either radians or degrees, according to the option you select with the OPTION statement. A nonzero text angle value causes text to be rotated from the normal orientation by the number of degrees or radians specified. Text rotation is in a counterclockwise direction; angles that are integer multiples of 180° display strings of characters horizontally and angles that are integer multiples of 90° display text vertically. Negative values cause the text to be rotated in a clockwise direction.

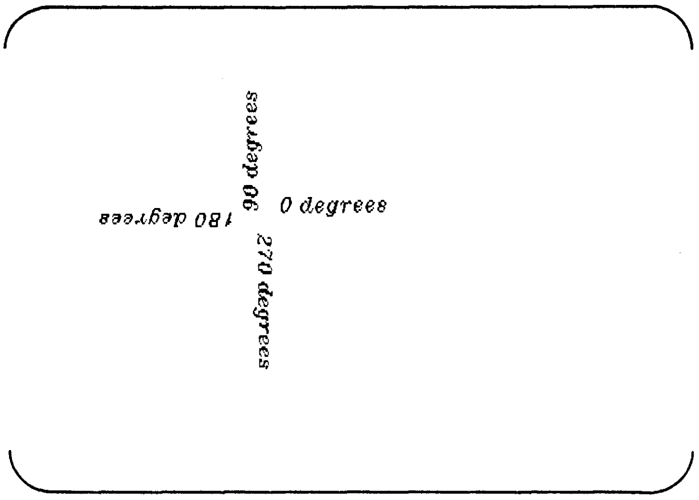
The following example illustrates the effect of the SET TEXT ANGLE statement:

Example

```
!+
!Set angle option to degrees
!-
OPTION ANGLE = DEGREES
SET TEXT HEIGHT 0.05
SET TEXT FONT -16, "STROKE"
SET TEXT ANGLE 90
GRAPH TEXT AT 0.5,0.5 : "90 degrees"
```

```
SET TEXT ANGLE 180
GRAPH TEXT AT 0.45,0.5 : "180 degrees"
SET TEXT ANGLE 270
GRAPH TEXT AT 0.5,0.45 : "270 degrees"
!+
!Reset to the default angle
!-
SET TEXT ANGLE 0
GRAPH TEXT AT 0.55,0.5 : "0 degrees"
END
```

Output



ZK-4914-86

The corresponding ASK TEXT ANGLE statement assigns the current valu of the text angle to a numeric variable you supply, for example:

```
OPTION ANGLE = DEGREES
DECLARE SINGLE turn
ASK TEXT ANGLE turn
```

6 Setting the Text Path

The normal text path places each character to the right of the previous character. The text in this manual is printed with a normal, or RIGHT, text path. Changing the text path is useful for labeling the axes in graphs. You can change the text path of the individual characters with the SET TEXT PATH statement. You supply a string expression for the path. You can enter the string in upper- or lowercase characters. Valid values for the string expression are

- RIGHT (the default)
- LEFT
- UP
- DOWN

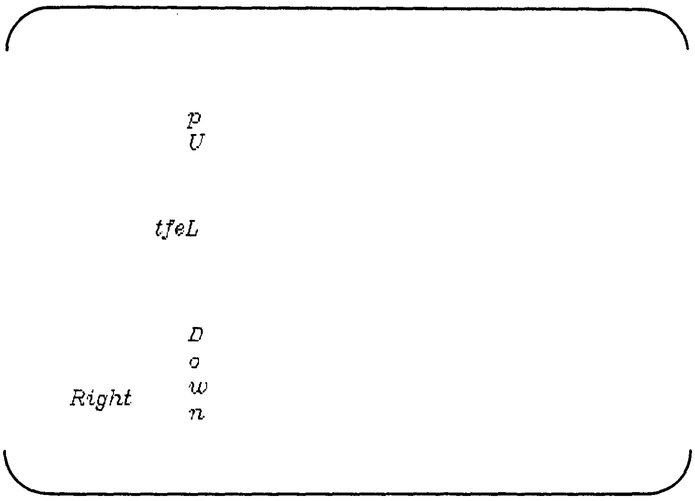
Characters are placed according to the value specified in the SET TEXT PATH statement as well as the current value of the text angle. The following example illustrates the four possible text paths using the default text angle:

Example

```
OPTION ANGLE = DEGREES
SET TEXT FONT -11 , "STROKE"
SET TEXT HEIGHT 0.05
!+
!Change the text path
!-
SET TEXT PATH "Up"
GRAPH TEXT AT 0.4,0.7 : "Up"

SET TEXT PATH "Left"
GRAPH TEXT AT 0.4,0.5 : "Left"
```

```
SET TEXT PATH "Down"
GRAPH TEXT AT 0.4,0.3 : "Down"
!+
!Reset path to default
!-
SET TEXT PATH "Right"
GRAPH TEXT AT 0.1,0.1 : "Right"
END
```

Output



ZK-4912-86

You can retrieve the current value of the text path with the correspondin ASK TEXT PATH statement. You supply this statement with a string variable for the path.

```
DECLARE STRING which_way
ASK TEXT PATH which_way
```

7 Setting the Text Justification

Text on a printed page is usually justified flush against the left margin, and is often justified on the right margin also. The text in this manual is justified only on the left margin. In VAX BASIC, you can justify text strings in a variety of ways.

The normal alignment of text strings varies with the direction of the text path, as shown in Table 4–1. For example, each of the words in this sentence follows a RIGHT text path with the horizontal justification at the left of each word and the left of each page. The vertical justification is at the base of each word.

Execution of a SET TEXT JUSTIFY statement allows you to alter the normal alignment of the text relative to the starting point you specify. This statement requires two arguments: string expressions for both the horizontal and vertical components of justification. The valid values for these string expressions are listed in Table 4–2. The values of NORMAL refer to the normal alignment as specified in Table 4–1. Following is an example of a SET TEXT JUSTIFY statement:

```
SET TEXT JUSTIFY "Left" , "Base"
```

Table 4–1: Normal Text Alignment

Text Path	Normal Horizontal Alignment	Normal Vertical Alignment
RIGHT	LEFT	BASE
LEFT	RIGHT	BASE
UP	CENTER	BASE
DOWN	CENTER	TOP

The *text starting point* refers to the x- and y-coordinates you supply with the GRAPH TEXT statement. For instance, in the following statement, the text starting point is 0.2,0.5.

```
GRAPH TEXT AT 0.2,0.5 :"Sample text"
```

Table 4–2: Text Justification Values

Component	Values	Explanation
Horizontal	NORMAL	Varies with the text path
	LEFT	Leftmost character placed at the text startin point
	CENTER	Center of the text string placed at the text starting point
	RIGHT	Text string placed so that the rightmost character in the text string ends at the text starting point
Vertical	NORMAL	Varies with the text path
	TOP	Top of the text string aligned with the text starting position
	CAP	Capline of the text string (highest point of the uppercase letters) aligned with the text starting position
	HALF	Starting point placed beginning an imaginar horizontal line through the middle of the te: string
	BASE	Baseline of the text string aligned with the starting point
	BOTTOM	Bottom of the text string (lowest point of letters such as p and q) aligned with the starting position

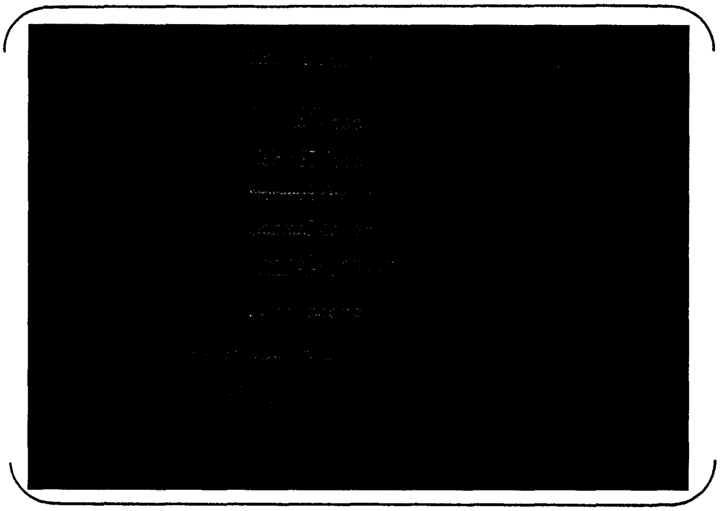
The following example illustrates many of the ways you can justify text. For purposes of illustration, the starting point for each text string is emphasized with a line beginning at the same point as the text string starting point. The example uses the default text angle of 0°. The first argument represents the value of the horizontal component of justificatior and the second argument represents the vertical component. Note that th values for NORMAL are dependent on the current text path, which in thi case is the default, RIGHT.

Example

```
SET TEXT HEIGHT 0.04
SET TEXT FONT -3 , "STROKE"
SET LINE COLOR 2

SET TEXT JUSTIFY "NORMAL", "NORMAL"
GRAPH TEXT AT 0.4,0.95 : "Normal/normal"
GRAPH LINES 0.4,0.95; 0.6,0.95

SET TEXT JUSTIFY "NORMAL", "TOP"
GRAPH TEXT AT 0.4,0.85 : "Normal/top"
GRAPH LINES 0.4,0.85; 0.6,0.85

SET TEXT JUSTIFY "NORMAL" , "CAP"
GRAPH TEXT AT 0.4,0.75 : "Normal/cap"
GRAPH LINES 0.4,0.75; 0.6,0.75

SET TEXT JUSTIFY "NORMAL" , "HALF"
GRAPH TEXT AT 0.4,0.65 : "Normal/half"
GRAPH LINES 0.4,0.65; 0.6,0.65

SET TEXT JUSTIFY "NORMAL" , "BASE"
GRAPH TEXT AT 0.4,0.55 : "Normal/base"
GRAPH LINES 0.4,0.55; 0.6,0.55

SET TEXT JUSTIFY "NORMAL" , "BOTTOM"
GRAPH TEXT AT 0.4,0.45 : "Normal/bottom"
GRAPH LINES 0.4,0.45; 0.6,0.45

SET TEXT JUSTIFY "LEFT" , "NORMAL"
GRAPH TEXT AT 0.4,0.35 : "Left/normal"
GRAPH LINES 0.4,0.35; 0.6,0.35

SET TEXT JUSTIFY "CENTER", "NORMAL"
GRAPH TEXT AT 0.4,0.25 : "Center/normal"
GRAPH LINES 0.4,0.25; 0.6,0.25
```

```
SET TEXT JUSTIFY "RIGHT", "NORMAL"
GRAPH TEXT AT 0.4,0.15 : "Right/normal"
GRAPH LINES 0.4,0.15; 0.6,0.15
END
```

Output



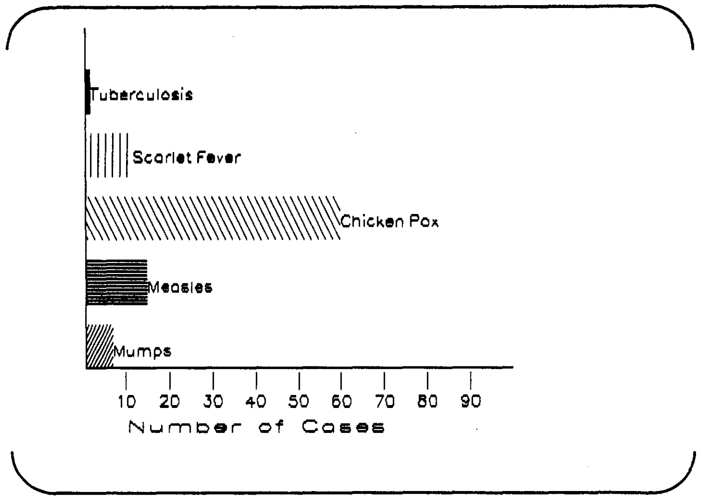
The following example changes the justification, path, and height of the text to label a simple bar chart. The chart shows the reported number of cases of communicable diseases in children in a small community during one year. The example includes the data within the program. Alternatively, if data is stored in a file, the file can be opened and read in the usual manner from within the program.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE start_y
DECLARE LONG loop
RECORD medical_record
      STRING disease
      SINGLE patients
END RECORD medical_record
DECLARE medical_record statistics(4)
```

```
!+
!Get the data
!-
DATA "Mumps",0.07,"Measles",0.15,                       &
     "Chicken Pox", 0.60,"Scarlet Fever",0.12,          &
     "Tuberculosis",0.02
FOR loop = 0 TO 4
  READ statistics(loop)::disease
  READ statistics(loop)::patients
NEXT loop
!+
!Draw the axes
!-
GRAPH LINES 0.01,1; 0.01,0.2; 1,0.2
!+
!Choose a new font
!-
SET TEXT FONT -3 , "STROKE"
SET TEXT HEIGHT 0.04
SET TEXT EXPAND 1.75
GRAPH TEXT AT 0.1,0.05: "Number  of  Cases"
SET TEXT JUSTIFY "CENTER" , "TOP"
SET TEXT EXPAND 1
GRAPH TEXT AT loop/10,0.2 : "|" FOR loop = 1 TO 10
GRAPH TEXT AT loop/100,0.15 : STR$(loop) FOR loop = 10 TO 90 STEP 10
!+
!Reset the text attributes to label bars
!-
SET TEXT PATH "RIGHT"
SET TEXT JUSTIFY "LEFT", "HALF"
!+
!Plot the data
!-
SET AREA STYLE "HATCH"
FOR loop = 0 TO 4
  start_y = 0.2 + (loop * 0.15)
  SET AREA STYLE INDEX INTEGER(start_y * 14)
  GRAPH AREA 0.01,start_y;                                       &
             statistics(loop)::patients,start_y;                 &
             statistics(loop)::patients,(start_y + 0.1);         &
             0.01,(start_y + 0.1)
  !+
  !Label the bars
  !-
  GRAPH TEXT AT statistics(loop)::patients,(start_y + 0.05)  &
              : statistics(loop)::disease
NEXT loop
END
```

Output



ZK-5508-86

You can retrieve the values of both the horizontal and vertical justificatior components by supplying string variables with the ASK TEXT JUSTIFY statement.

```
ASK TEXT JUSTIFY which_horiz , which_vert
```

4.8 Asking About Text Dimensions

You can think of any graphics text string as being surrounded by an imaginary box. Many characters extend beyond the box in one or more directions, while other characters are defined within the perimeter of the box. As you alter the height, width, and angle of characters, you also alter the dimensions and positioning of the text box. Although you know the starting position of the text string, it is often difficult to predict where the text string will end. If you know where the text string ends, you know where to start a subsequent text string (for instance, if you want to concatenate strings).

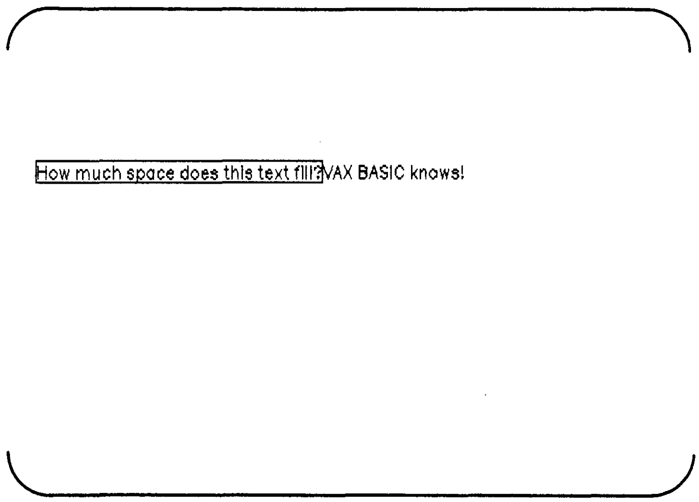
You can retrieve the dimensions of this text box for any text string with the ASK TEXT EXTENT statement. You supply the statement with the text string and the starting point, as well as two arrays to retrieve the coordinates of the corner points of the box.

If you want to attach an additional text string, you need to know the coordinates of the nearest point where subsequent text can be added. The coordinates for this concatenation point can be retrieved with the ASK TEXT POINT statement. The following example illustrates both the ASK TEXT EXTENT and the ASK TEXT POINT statements:

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE concat_x, concat_y
DECLARE STRING CONSTANT text_string =                    &
              "How much space does this text fill?"
DIM SINGLE x_coords(3), y_coords(3)

SET TEXT FONT -3 , "STROKE"
SET TEXT HEIGHT 0.04
ASK TEXT EXTENT , text_string                            &
                AT 0.0,0.6                               &
                : x_coords, y_coords
ASK TEXT POINT  , text_string                            &
                AT 0.0,0.6                               &
                : concat_x, concat_y
!+
!Display the text
!-
GRAPH TEXT AT 0.0,0.6 : text_string
!+
!Use the text box coords to draw the text box
!by graphing a hollow area
!-
SET AREA STYLE "Hollow"
MAT GRAPH AREA x_coords , y_coords
!+
!Use the concatenation point to add on a string
!-
GRAPH TEXT AT concat_x, concat_y : "VAX BASIC knows!"
END
```

Output



ZK-4907-86

For another example using the ASK TEXT POINT statement, see the SET TEXT COLOR statement in Chapter 9.

4.9 Summary

This chapter has explained how to use the following statements:

- SET/ASK TEXT FONT
- SET/ASK TEXT HEIGHT
- SET/ASK TEXT ANGLE
- SET/ASK TEXT SPACE
- SET/ASK TEXT EXPAND
- SET/ASK TEXT PATH
- SET TEXT JUSTIFY
- ASK TEXT EXTENT
- ASK TEXT POINT

The examples so far in this manual have all placed graphics objects in the default drawing board, where 0,0 is a point in the bottom corner of the terminal screen. It is often desirable to change the scale and orientation of the coordinate plane. The following chapter shows you how to alter the drawing board to suit your own needs.

Chapter 5

The Drawing Surface

Using the default drawing board allows you to create simple graphics on your screen quickly and easily. At the same time, using the default limits flexibility in programming techniques and the format of screen displays. This chapter shows you how to define and extend the default drawing board to suit your own application and describes how to:

- Change the window setting
- Define transformations
- Clip images

1 Setting the World Window

Whether you have plotted points on paper with a grid or plotted points in your mind's eye, so far you have expressed the position of these points within the boundaries of the default drawing board. The default drawing board is also known as the *world window*. The coordinates that you use to define the points in the world window are known as *world coordinates*.[1]

The boundaries of the default world window are 0,1,0,1 for the left, right, bottom, and top edges respectively. The world coordinates of the points you have plotted have been contained within these same limits; that is, the x- and y-coordinates of points in the default world window have values ranging from 0 through 1. It is not always convenient or suitable to restrict the coordinates of points to values from 0 through 1, nor is it necessary. You can change the boundaries of the world window and the range of world coordinate values of points that can be displayed on your screen.

[1] Another common term for world coordinates is *problem coordinates*.

You can set the boundaries of the world window to your own preferenc
For example, window boundaries of -20,20,-20,20 allow you to plot poir
with negative as well as positive coordinates and accommodate 0,0 as th
center point. Boundaries need not be symmetrical; you can define the
x-axis as being, for example, 0 to 50, with a y-axis of 40 to 240.

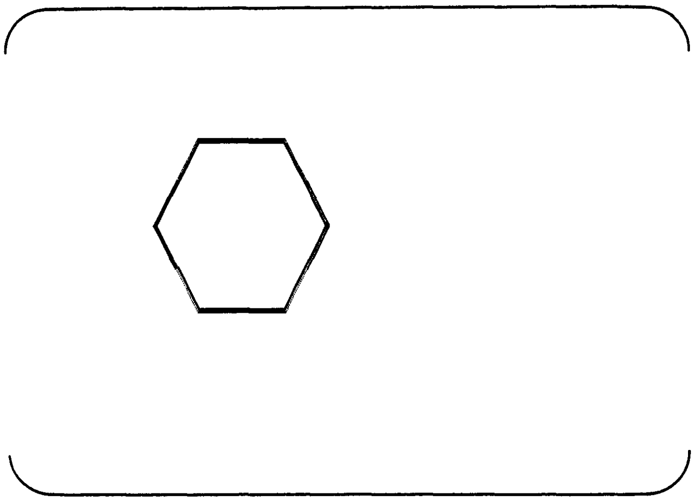
The boundaries you set merely reflect a scale. Whatever boundaries you
select, VAX BASIC still displays graphics images on the same area on yo
screen, unless you make further changes as described in later sections o
this chapter.

To set new boundaries for your world window, use the SET WINDOW
statement. The following example sets the world window boundaries to
be 0,100,0,100. The program displays a hexagon; the coordinates of the
points of the hexagon can now range from 0 to 100 because of the new
window boundaries.

Example

```
DECLARE LONG loop, SINGLE x, y
!+
!Set the world window
!-
SET WINDOW 0,100,0,100
!+
!Express coordinates in range of 0 to 100
!-
DATA 70,50,60,30,40,30,30,50,40,70,60,70,70,50
!+
!Draw hexagon
!-
SET LINE SIZE 5
FOR loop = 0 TO 6
   READ x,y
   PLOT LINES x, y;
NEXT loop
END
```

Output



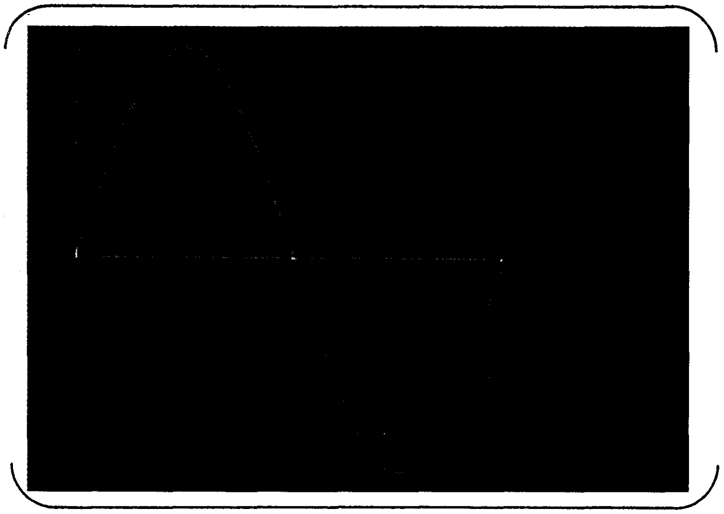
ZK-4917-86

In Chapter 2, an adjustment was made to the formula that produces the coordinate points for the sine curve. The adjustment was necessary because some values fall outside the default world window boundaries. When world window boundaries are set to accommodate negative values, this adjustment is no longer necessary. The following example sets the world window boundaries to 0,2*PI,-1,1. These boundaries accommodate the coordinate values of all points on the sine curve. This example also draws the axes.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE loop_index, x, y
!+
!Set the window boundaries
!-
SET WINDOW 0, 2*PI, -1, 1
!+
!Draw in the axes
!-
PLOT LINES 0,0; 2*PI,0  !horizontal axis
PLOT LINES 0,-1; 0,1  !vertical axis
!+
!Plot the coordinates of points on sine curve
!-
SET LINE SIZE 3
SET LINE COLOR 2
FOR loop_index = 0 TO 2*PI STEP 0.1
    PLOT LINES loop_index, SIN(loop_index);
NEXT loop_index
PLOT LINES (2*PI + 0.1), SIN(2*PI + 0.1)
END
```

Output



When you set the window, you may also need to change the height of th text. The default text height is 0.035 world coordinate units. After you set new boundaries with the SET WINDOW statement, the proportion of the text height in relation to the height of the world window changes. To maintain a text height that is equivalent to the default height, use the following formula:

new_text_height = max_height − min_height ∗ 0.035

Example

```
SET WINDOW 0,10,500,800
new_text_height = (800 - 500) * 0.035
SET TEXT HEIGHT new_text_height
```

Notice the values passed to the SET TEXT HEIGHT statement in the following example. The world window height is set from 0 to 2000; therefore, the default text height of 0.035 is no longer appropriate. The example alters the text height to a height suitable for the display.

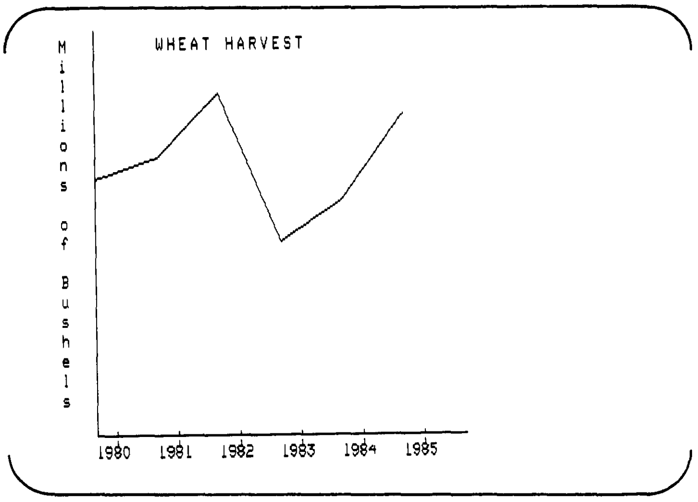
When you plot values on a graph, you can use the axes as measures of the values you are working with. You can set the world window to directly correspond to the values you are plotting. The following example sets the world window boundaries to correspond to the years 1979 through 1986, and to the wheat yield in millions of bushels. The graph shows only the years under study. The axes are drawn well within the window boundaries to leave room for explanatory text on the axes.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG year
DIM LONG bushels(1980 TO 1985)
!+
!Set window boundaries to suit data
!-
SET WINDOW 1979,1986,0,2000
!+
!Initialize array with bushels per year data
!-
bushels(1980) = 1300
bushels(1981) = 1400
bushels(1982) = 1700
bushels(1983) = 1000
bushels(1984) = 1200
bushels(1985) = 1600
```

```
!+
!Draw the axes
!-
GRAPH LINES 1980,2000; 1980,100; 1986,100
!+
!Plot the coordinates for bushels of wheat
!-
PLOT LINES : year,bushels(year); FOR year = 1980 TO 1985
!+
!Set text font & height for headers
!-
SET TEXT FONT -1, "CHAR"
SET TEXT HEIGHT 60
GRAPH TEXT AT 1981,1900 : "WHEAT HARVEST"
!+
!Graph text for vertical axis
!-
SET TEXT PATH "DOWN"
GRAPH TEXT AT 1979.5,1950 : "Millions of Bushels"
!+
!Graph text for horizontal axis
!-
SET TEXT PATH "RIGHT"
SET TEXT JUSTIFY "LEFT" , "TOP"
GRAPH TEXT AT (year + 0.25),120 : "|" FOR year = 1980 TO 1985
!+
!Place markers on axis for years
!-
SET TEXT FONT -1, "STROKE"
SET TEXT JUSTIFY "CENTER" , "NORMAL"
GRAPH TEXT AT (year + 0.25),5 : NUM$(year)  FOR year = 1980 TO 1985
END
```

Output



ZK-5511-86

A complementary ASK WINDOW statement allows you to retrieve the values of the world window boundaries. For example, the following statement returns the implied boundary values in the variables *left_1, right_1, bottom,* and *top.* These values can be used to reset the world window at a later point in a program, or to calculate a suitable text height.

Example

```
DECLARE SINGLE left_1, right_1, bottom, top
    .
    .
    .
ASK WINDOW left_1,right_1,bottom,top
!+
!Set text height to 1/10th of current window height
!-
SET TEXT HEIGHT ((top - bottom)/10)
    .
    .
    .
```

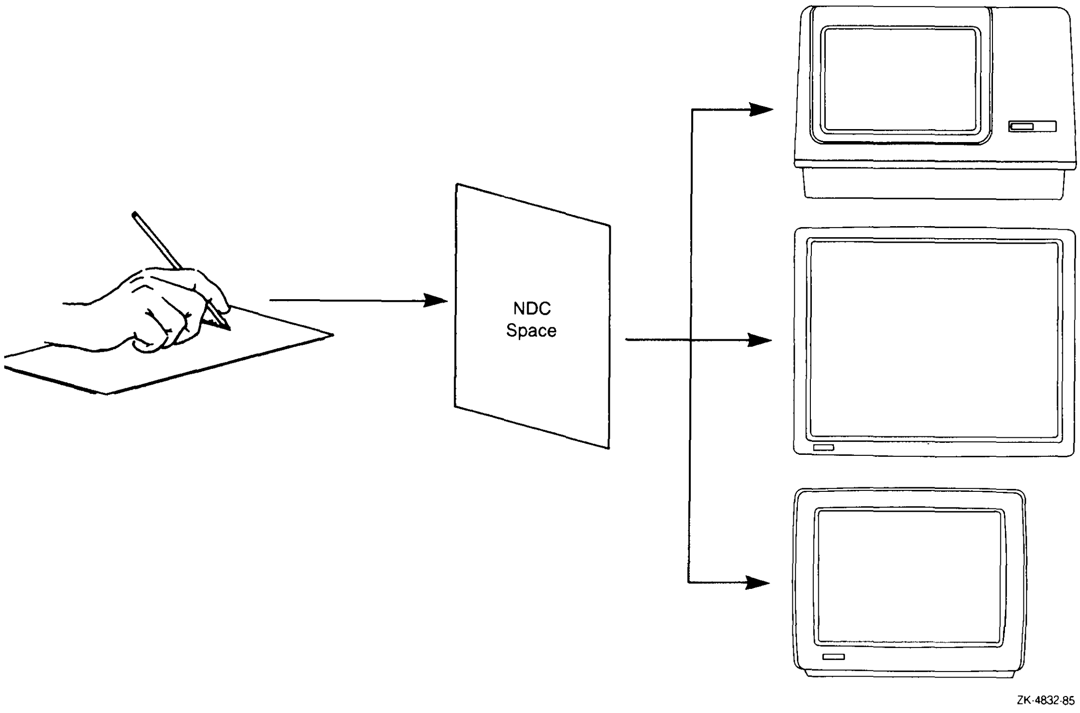
Both the SET WINDOW and ASK WINDOW statements accept an optional parameter for the *transformation* number. Transformations are explained in the following section.

5.2 Defining a Transformation

So far, the images you have plotted on your drawing surface seem to be mapped directly onto the default drawing board on your screen. Howeve your screen can be any supported graphics device. Considerable work is done "behind the scenes" to make the image you plot portable to any graphics output device. In fact, the image you define in the world windo must go through a *transformation* to be displayed as output on any one o the devices supported by VAX GKS.

A transformation is the procedure VAX BASIC performs on an image defined in a window so that the image can be displayed on one of the supported devices. Figure 5–1 illustrates the concept of this internal processing by showing how one image defined in world coordinates is displayed on different output devices.

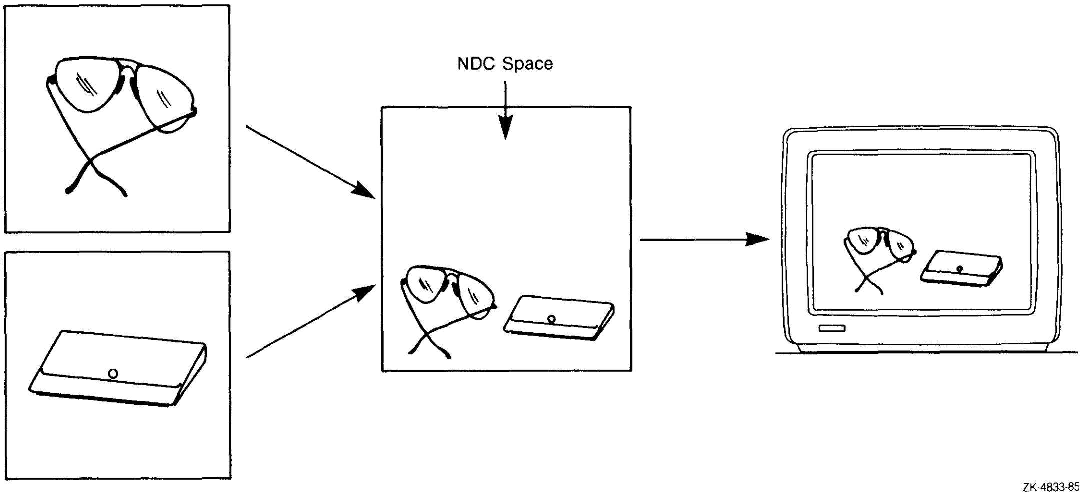
igure 5–1: Transformations Let You Display Images on Many Supported Devices



The transformation occurs in an abstract square coordinate plane known as the *Normalized Device Coordinate* (NDC) space. VAX BASIC maps the image you define onto NDC space; this is known as a *normalization* transformation. The image is then mapped to the output device; this is known as the *device* transformation. You can alter the internal processing of images to define your own transformations and thereby gain increased flexibility in your graphics displays. This chapter describes only normalization transformations; device transformations are explained in Chapter 8.

NDC space has boundaries that are always 0,1,0,1 for the left, right, bottom, and top boundaries respectively. You can select portions of NDC space in which to position your image, you can position several images onto NDC space, and you can select which portion of NDC space to map to your screen. Figure 5–2 illustrates how NDC space can be used to build up a screen display. Separate images can be defined in windows with different boundaries, projected onto different portions of NDC space and displayed as one composite image on the screen. This figure also illustrates the progression of any images from your drawing surface to NDC space, and from NDC space to your screen.

Figure 5–2: Creating a Composite Image

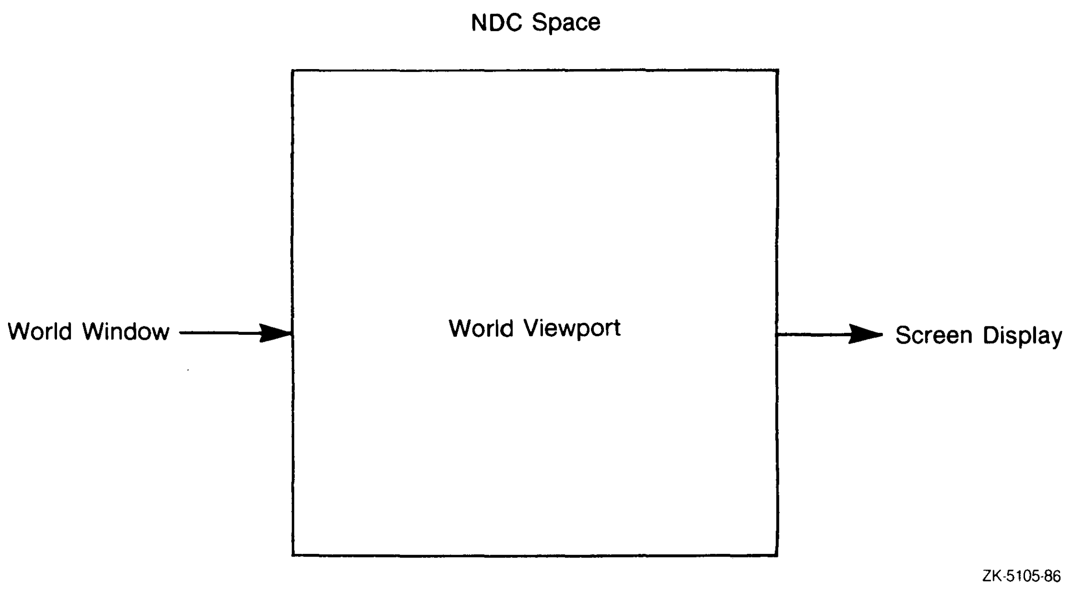


A transformation is defined using windows and viewports. A *window* is where an image is defined. A *viewport* is where an image is displayed or projected. The image defined in a window is projected onto a viewport. The progression of a transformation can be seen as a two-part process:

1. An image is defined in the world window.
2. The image in the world window is projected onto the world viewport in NDC space.

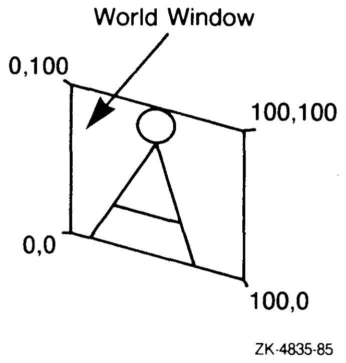
When you change the world window or world viewport defaults, you are defining a new transformation. This process is illustrated in Figures 5–3, 5–4, and 5–5.

igure 5–3: The Transformation of an Image Through NDC Space



You alter the boundaries of the world window with the SET WINDOW statement, as shown in Section 5.1. The default world window boundaries are 0,1,0,1. Figure 5–4 illustrates a world window of 0,100,0,100; coordinate values of points in this window can range from 0 through 100.

Figure 5–4: Defining an Image in a World Window



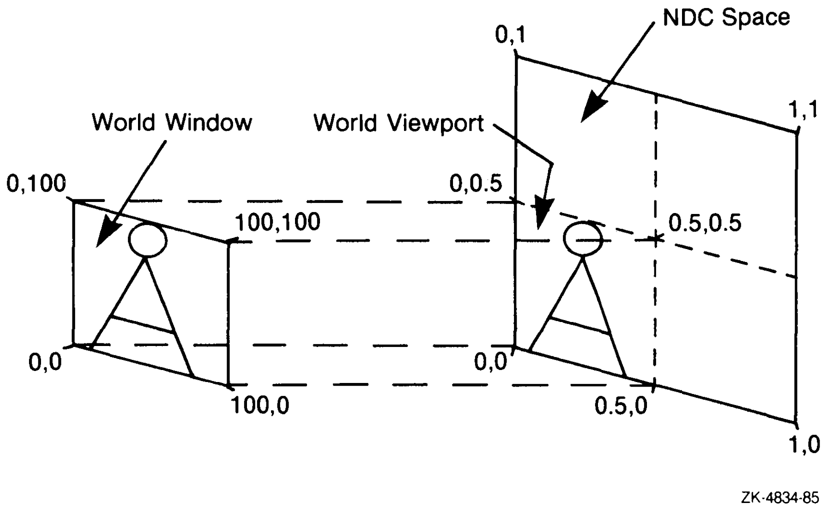
The image defined in the world window or drawing surface is projected onto the world viewport in NDC space, as shown in the following section

5.3 Setting the World Viewport

Each of the examples in Section 5.1 changes the boundaries for the world window without making any further changes. The images are still mappec to the default viewport on NDC space. You can select portions of NDC space and create more than one world viewport.

Using the wheat yield example, you can place the same graph in the lowei left corner of NDC space, rather than placing it in all of NDC space. The default world viewport is all of NDC space, with boundaries of 0,1,0,1. The SET VIEWPORT statement allows you to set new boundaries for the world viewport. You can use either all of NDC space or a portion of it: you cannot use boundary values beyond the range of 0 through 1. Figure 5–5 illustrates a world viewport of 0,0.5,0,0.5.

Figure 5–5: Projecting an Image onto a World Viewport

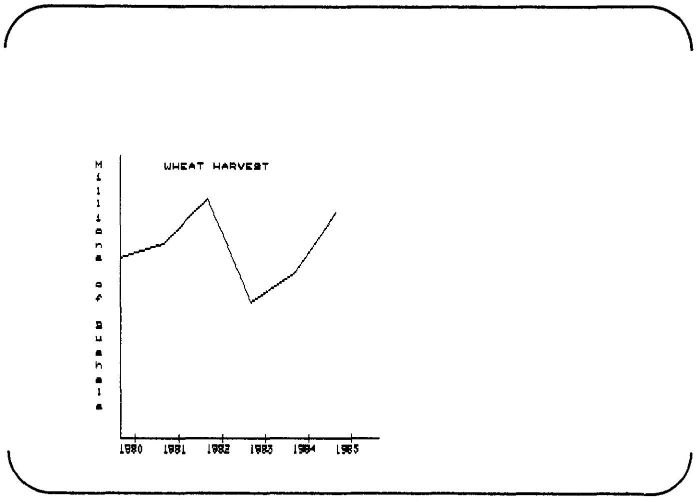


The following statement limits the world viewport to the lower left corner of NDC space. If no other changes are made, the addition of this one statement prior to any graphics output statements in the program causes the line chart to be displayed in the lower left corner of the display area.

Example

```
    .
    .
    .
SET VIEWPORT 0,0.7,0,0.7
!+
!Continue with output of wheat yield data
!-
    .
    .
    .
```

Output



ZK-5512-86

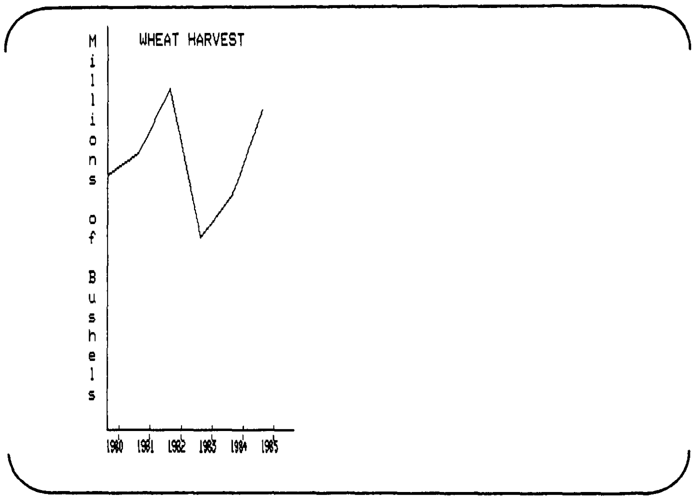
Notice that a square diagram has been transformed to a square portion of NDC space. When the window and viewport shapes do not match, the proportions are distorted. The ratio of one scale to another is known as the *aspect ratio*. When the aspect ratio is not 1:1, distortion will be apparent to some degree.

The following display demonstrates the scaling problems that can occur. In this case a square world window is transformed to a rectangular world viewport. The mismatch of shapes results in a distortion of the scales. The aspect ratio is discussed further in Chapter 8 in relation to device transformations.

Example

```
    .
    .
    .
!+
!Change the viewport for the default transformation
!-
SET VIEWPORT 0,0.5,0,1
!+
!Continue with wheat/year data as before
!-
    .
    .
    .
```

Output



ZK-5513-86

You can retrieve the boundaries of the current viewport with the ASK VIEWPORT statement. For example:

```
ASK VIEWPORT left_1,right_1,bottom,top
```

5.4 Multiple Transformations

Several images can each be placed in different sections of NDC space, and then all of NDC space displayed on the screen. Each image that requires a different window or viewport requires a transformation; this opens up the potential for confusion between one transformation and another. To keep track of the window and viewport you select for a particular display, you can identify each transformation with a number. The transformation number must be an integer from 1 to 255. Once a particular transformation has been defined, you can refer to it again by the same number.

Transformations can be optionally specified in the SET WINDOW and SET VIEWPORT statements explained earlier. The default transformation is 1. At the start of program execution, transformation 1 consists of a window and viewport with boundaries of 0,1,0,1. You can redefine transformation 1 to your choice of window and viewport boundaries.

If you do not specify a transformation number, VAX BASIC uses the default transformation of 1. A statement such as the following sets up the new window boundaries for transformation 1, because no other transformation is specified.

```
SET WINDOW 0,100,0,100
```

In the examples so far in this chapter, only one transformation has been defined within a program. When several transformations are defined in a program, you should specify the transformation number in SET VIEWPORT and SET WINDOW statements.

```
SET VIEWPORT , TRAN 2 : 0.5,1,0.5,1
```

Only one transformation is current for displaying points at a given time during execution. You can specify which transformation is the current one. Otherwise, the last transformation you set in your program remains the current transformation. The current transformation is explicitly established with a SET TRANSFORMATION statement, for example:

```
SET TRANSFORMATION 2
```

You can implicitly establish the current transformation with a SET WINDOW or SET VIEWPORT statement. For instance, the following statement implicitly sets the current transformation to 7. The window boundaries for any other transformations you may have defined remain intact and unaffected by this SET VIEWPORT statement.

```
SET VIEWPORT, TRAN 7 : 0,0.5,0,0.5
```

Subsequent graphics output statements use transformation 7 until VAX BASIC executes another SET TRANSFORMATION, SET WINDOW, or SET VIEWPORT statement. The ASK TRANSFORMATION statement allows you to retrieve the number for the transformation that is current, for example:

```
DECLARE LONG current
ASK TRANSFORMATION current
```

In the following example, transformation 1 is redefined and an additional transformation, 2, is defined. A SET TRANSFORMATION statement sets the current transformation to 1 for subsequent graphics output.

Example

```
!Tran 1 becomes the current transformation
SET WINDOW , TRAN 1 : 50,100,0,250

!Tran 2 now becomes the current transformation
SET WINDOW , TRAN 2 : -5,5,-5,5
    .
    .
    .
!Tran 1 now becomes the current transformation
SET TRANSFORMATION 1
GRAPH AREA : 50,0; 50,250; 100,250; 100,0
    .
    .
    .
```

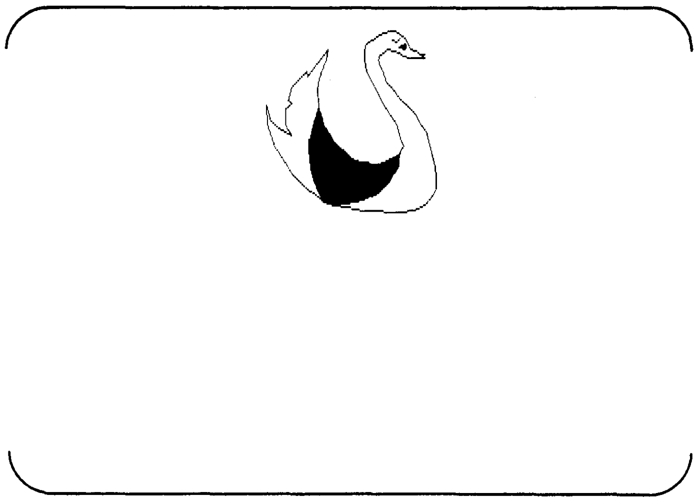
The following example displays the image of a swan using the transformation identified as 1. Notice that the transformation number is also referred to in the SET WINDOW and SET VIEWPORT statements. In this example, only one transformation is defined. The data provided in this picture is used in examples in later chapters of this manual. The image of the swan is transformed to the top right corner of NDC space.

Example

```
PROGRAM Draw_swan
  EXTERNAL PICTURE swan
  SET WINDOW , TRAN 1 : 0,100,0,100
  SET VIEWPORT , TRAN 1 : 0.5,1,0.5,1
  DRAW swan
END PROGRAM
```

```
PICTURE swan
 OPTION TYPE = EXPLICIT
 DECLARE LONG counter
 DIM SINGLE x_array(98),y_array(98),                                      &
            mark_x(3),mark_y(3),                                          &
            eye_x(2),eye_y(2),                                            &
            wing_area_x(28),wing_area_y(28)
 Outline_data:
  DATA 82,80.5,80,80.5,75,81,72,81.5,70,83.5,68,84,65,84.5,               &
  63,83,61,82,60.5,80,60.55,78,62.5,75,64,70,66.5,67,70,62.5,             &
  72,60,75,57,78,54,80,50,82.5,46,84,40,86.5,30,87,24.5,87,20,            &
  86,17.5,83.5,13.5,82,12,80,11.5,77,10,70,9,65,9,60,9.2,53,10,           &
  50,10.5,40,12,35,14,30,18,27,20,20,27.5,14,36,12,40,8.5,50,8,           &
  59,10,52.5,14,48,20,45,17,50,16.75,57,20,60,18,61.5,20,67,27,           &
  75,27.5,72,37,85,36,80,34,75,32.25,70,32,65,32.25,60,32.35,             &
  58,34,53,35,50,38,45,40,41,43.5,38,48,34,52,32.5,60,31,63,32,           &
  69.5,35.5,70.5,37,72,40,70.5,43,69.5,47,68.5,50,65.5,55,62.5,           &
  60,60,65,57.5,70,55,75,54,80,54.25,83,54,84.5,55,87,59,90,60,           &
  91,63,92.25,66.5,93,68,92.75,70,92,71.5,91,73,89.5,74,87,75.5,          &
  85,78,84,80,82,82,82,80,80.5,82,80.5

 Wing_area_data:
  DATA 32.35,58,34,53,35,50,38,45,40,41,43.5,38,48,34,52,32.5,            &
  60,31,63,32,69.5,35.5,70.5,37,32.35,58,30.05,50,28,42,27.5,             &
  44.5,28.5,30,30,24,31.5,20,35,14,40,12,45,13,50,14,53,15,               &
  58.5,17.5,61,19,64.5,22,69,30,70.5,37

 Head_mark_data:
  DATA 73,87,70,86,73,85,73.15,86,67,89,69,88,70.5,90

 READ x_array(counter),y_array(counter) FOR counter = 0% TO 98%
 READ wing_area_x(counter),wing_area_y(counter) FOR counter = 0% TO 28%
 READ mark_x(counter),mark_y(counter) FOR counter = 0% TO 3%
 READ eye_x(counter), eye_y(counter) FOR counter = 0% TO 2%

 Display_the_swan:
  SET LINE COLOR 3%
  SET LINE STYLE 1
  MAT GRAPH LINES x_array,y_array
  SET AREA STYLE "PATTERN"
  MAT GRAPH AREA wing_area_x,wing_area_y
  SET LINE COLOR 2%
  MAT GRAPH LINES eye_x,eye_y
  MAT GRAPH AREA mark_x,mark_y
END PICTURE
```

Output


ZK-4916-86

You can also specify a transformation in the ASK WINDOW and ASK VIEWPORT statements, for instance:

```
ASK WINDOW , TRAN 4 : left_1,right_1,bottom,top
```

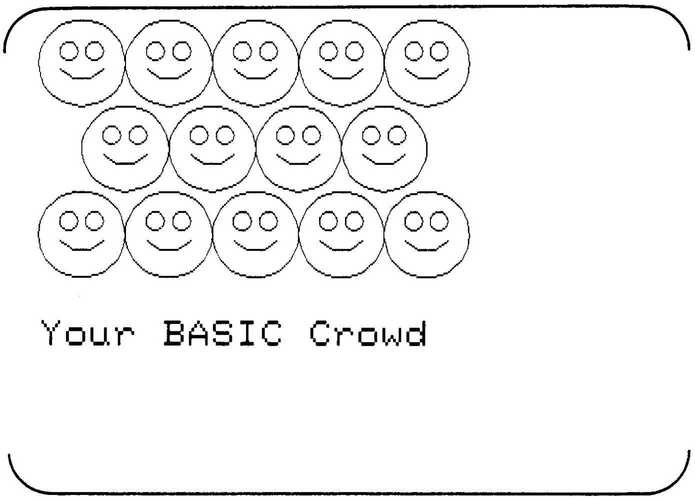
Use of these statements is further illustrated in the following example, which transforms one face onto fourteen different portions of NDC space. Adding some text makes a total of fifteen transformations. Without changing the current transformations, all of the faces would have been placed directly over the first in the top left section of NDC space. This example uses a simple picture called *face*. Notice that the transformations are defined before each picture invocation. Chapter 6 provides details governing the use of pictures.

Example

```
PROGRAM draw_crowd
EXTERNAL PICTURE face
!+
!Select the default world window for all transformations
!-
SET WINDOW , TRAN loop : 0,1,0,1     FOR loop = 1 TO 15
!+
!Set up fourteen different world viewports in NDC space
!-
SET VIEWPORT , TRAN 1 :  0,   0.2, 0.8, 1
SET VIEWPORT , TRAN 2 :  0.2, 0.4, 0.8, 1
SET VIEWPORT , TRAN 3 :  0.4, 0.6, 0.8, 1
SET VIEWPORT , TRAN 4 :  0.6, 0.8, 0.8, 1
SET VIEWPORT , TRAN 5 :  0.8, 1,   0.8, 1
SET VIEWPORT , TRAN 6 :  0.1, 0.3, 0.6, 0.8
SET VIEWPORT , TRAN 7 :  0.3, 0.5, 0.6, 0.8
SET VIEWPORT , TRAN 8 :  0.5, 0.7, 0.6, 0.8
SET VIEWPORT , TRAN 9 :  0.7, 0.9, 0.6, 0.8
SET VIEWPORT , TRAN 10 : 0,   0.2, 0.4, 0.6
SET VIEWPORT , TRAN 11 : 0.2, 0.4, 0.4, 0.6
SET VIEWPORT , TRAN 12 : 0.4, 0.6, 0.4, 0.6
SET VIEWPORT , TRAN 13 : 0.6, 0.8, 0.4, 0.6
SET VIEWPORT , TRAN 14 : 0.8, 1,   0.4, 0.6
!+
!Set up the world viewport for the text
!-
SET VIEWPORT , TRAN 15 : 0,1,0,1
!+
!Draw a face in each world viewport
!-
FOR tran_id = 1 TO 14
    SET TRANSFORMATION tran_id
    DRAW face
NEXT tran_id

Write_the_text:
   SET TRANSFORMATION 15
   SET TEXT HEIGHT 0.08
   GRAPH TEXT AT 0,0.2 : "Your BASIC Crowd"
END PROGRAM
```

```
PICTURE face
!+
!Set up one image of the face on the world window
!-
OPTION TYPE = EXPLICIT
DECLARE SINGLE turn,increment,                   &
             head_x(40),head_y(40),              &
             mouth_x(6),mouth_y(6),              &
        LONG loop, counter,                      &
                tran_id
DECLARE SINGLE CONSTANT radius = 0.5
DECLARE LONG CONSTANT npoints = 40
Head:
   increment = 2 * PI/npoints
   turn = 0
   FOR loop = 0 TO npoints -1
       head_x(loop) = COS(turn) * radius + 0.5
       head_y(loop) = SIN(turn) * radius + 0.5
       turn = turn + increment
   NEXT loop
   head_x(40) = head_x(0)
   head_y(40) = head_y(0)
Mouth:
   DATA 0.25,0.45,0.34,0.36,0.4,0.325,0.5,       &
        0.324,0.6,0.325,0.66,0.365,0.75,0.425
   READ mouth_x(loop),mouth_y(loop) FOR loop = 0% TO 6%
Draw_face:
     MAT GRAPH LINES head_x,head_y
     MAT GRAPH LINES mouth_x,mouth_y
     !Eyes
     SET POINT STYLE 4
     GRAPH POINTS 0.35,0.65; 0.65,0.65
END PICTURE
```

Output



ZK-4920-86

5.5 Clipping the Image

The boundaries of the world window are also the limits of the world coordinates that are displayed on your screen. If you plot points beyond the limits of the world window you have defined, these points are also beyond the limits of the world viewport and are not displayed by default. This is known as *clipping*. Any points that are beyond the boundaries of the world window are clipped by default. VAX BASIC clips images at the boundaries of the world viewport. Clipping affects all graphics objects: lines, points, areas, and text.

You can control the clipping feature with the SET CLIP statement. The SET CLIP statement accepts string expressions equivalent to "ON" or "OFF". The default clipping status is ON. When you turn clipping off, VAX BASIC displays graphics objects that are beyond the boundaries of the world window but are still within the boundaries of NDC space.[2]

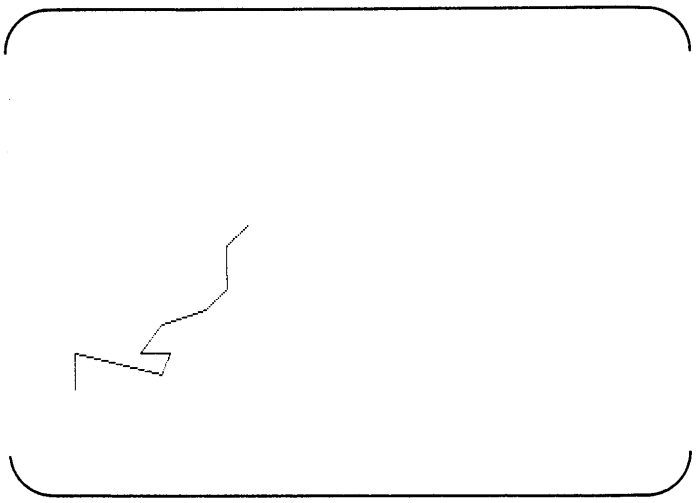
[2] The entire NDC space is also the default device window. When the device window is altered and clippin is turned off, objects that are beyond the boundaries of the world window but are within the boundarie of the device window are displayed. For more information about device transformations, see Chapter 8.

The following example defines a kite and string. Only the string is plotted within the world window; the points for the body of the kite are plotted in coordinates that are beyond the extent of the world window. With clipping on (the default), only the kite string is displayed.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG loop
DIM SINGLE kite_x(19),kite_y(19)
SET WINDOW , TRAN 1 : 0,50,0,50
SET VIEWPORT , TRAN 1 : 0,0.5,0,0.5
!+
!Get the data for the kite
!Some points are beyond the world window
!-
DATA 80,90, 90,70, 50,50, 80,90,              &
     50,80, 90,70, 50,80, 50,50,              &
     50,50, 45,45, 45,40, 45,35,              &
     40,30, 34,28, 30,27, 25,20,              &
     32,20, 30,15, 10,20, 10,12
READ kite_x(loop),kite_y(loop) FOR loop = 0% TO 19%
!+
!Draw the kite
!-
MAT GRAPH LINES kite_x, kite_y
GRAPH POINTS 68,82; 77.5,79.5
```

Output



ZK-4915-86

To display the whole kite, you must change the clipping status with the following statement:

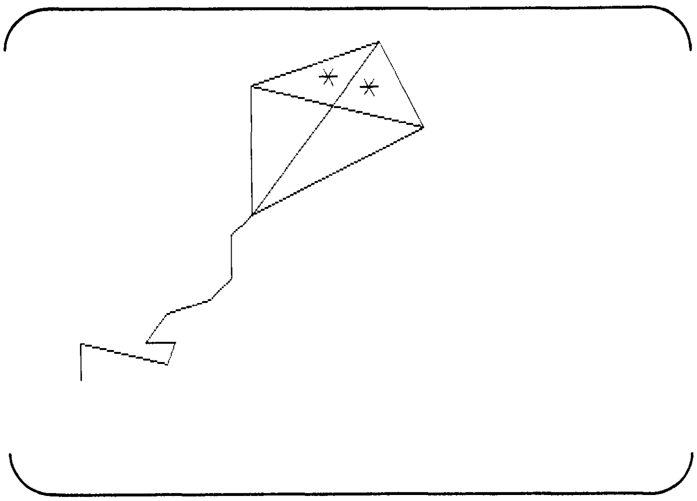
```
SET CLIP "OFF"
```

Adding this statement to the previous example turns off the clipping and displays both the kite and the string:

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG loop
DIM SINGLE kite_x(19),kite_y(19)
SET WINDOW , TRAN 1 : 0,50,0,50
SET VIEWPORT , TRAN 1 : 0,0.5,0,0.5
DATA 80,90, 90,70, 50,50, 80,90,                &
     50,80, 90,70, 50,80, 50,50,                &
     50,50, 45,45, 45,40, 45,35,                &
     40,30, 34,28, 30,27, 25,20,                &
     32,20, 30,15, 10,20, 10,12
READ kite_x(loop),kite_y(loop) FOR loop = 0% TO 19%
!+
!Turn off clipping
!-
SET CLIP "OFF"
!+
!Draw the kite
!-
MAT GRAPH LINES kite_x, kite_y
GRAPH POINTS 68,82; 77.5,79.5
END
```

Output



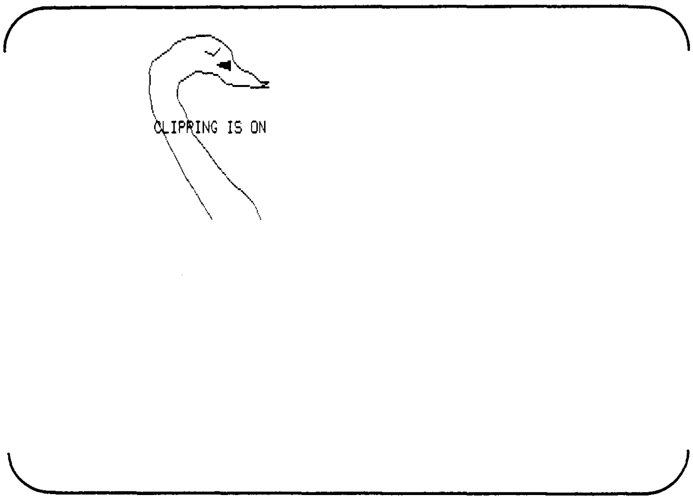
ZK-4923-86

In the following example, the world window is set to 50,100,50,100. However, the coordinates for the points of the swan range from 0 through 100. All points that extend beyond the world window are cut off, or clipped, in the first display because the clipping status is on by default. Clipping is turned off for the second screen; therefore, all of the swan is displayed.

Example

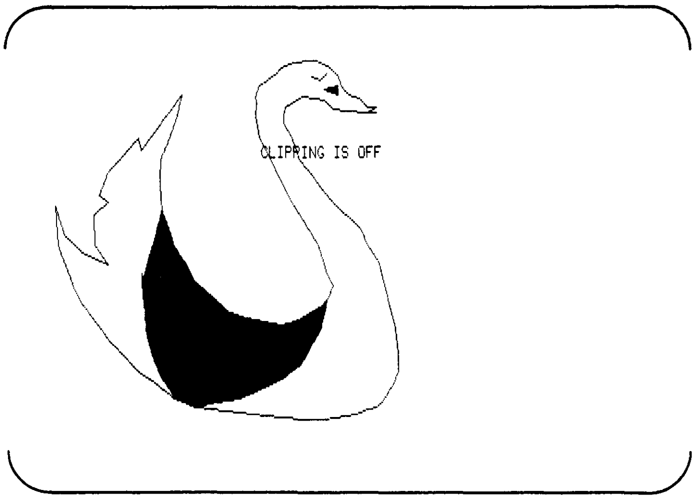
```
EXTERNAL PICTURE swan
!+
!Set up points for swan in world window
!Coordinates in picture are spread from 0 through 100
!Half of swan is out of the world window
!-
SET WINDOW , TRAN 1 : 50,100,50,100
SET VIEWPORT , TRAN 1 : 0.5,1,0.5,1
SET TEXT HEIGHT 2.5
    .
    .
    .
!+
!Leave default clipping
!Display screen 1
!-
GRAPH TEXT AT 55,70 : "CLIPPING IS ON"
DRAW swan
SLEEP 5%
CLEAR
    .
    .
    .
!+
!Turn off clipping
!-
SET CLIP "OFF"
!+
!Display screen 2
!-
GRAPH TEXT AT 55,70 : "CLIPPING IS OFF"
DRAW swan
END
```

Output Screen 1:



ZK-4988-86

Output Screen 2:



ZK-4985-86

In the following example, two fuses are presented without the text labels when clipping is on for the first screen. For the second screen, the clippir is turned off; therefore, the labels are displayed.

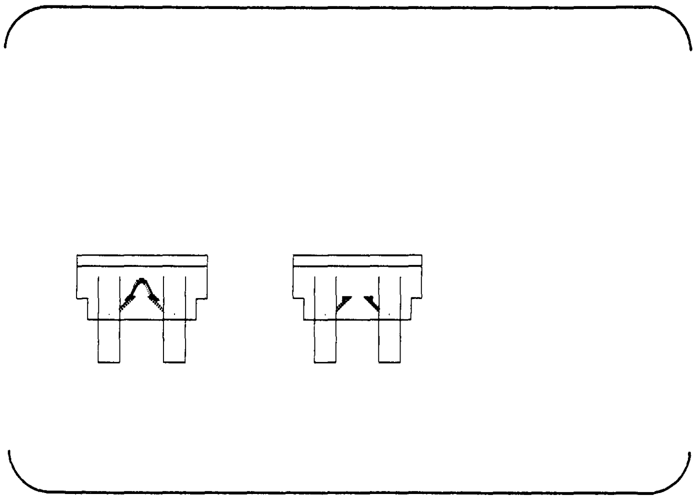
Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE fuse_shell,good_fuse,bad_fuse
SET WINDOW , TRAN 1 : 0,100,0,100
SET VIEWPORT , TRAN 1 : 0,0.5,0,0.5
SET TEXT COLOR 2
SET TEXT HEIGHT 6
!+
!Draw good fuse
!-
GRAPH TEXT AT 30,110 : "GOOD FUSE"
DRAW fuse_shell
DRAW Good_fuse
!+
!Draw bad fuse
!-
SET WINDOW , TRAN 2 : 0,100,0,100
SET VIEWPORT , TRAN 2 : 0.5,1,0,0.5
GRAPH TEXT AT 30,110 : "BAD FUSE"
DRAW fuse_shell
DRAW Bad_fuse
SLEEP 5%
CLEAR
!+
!Turn off clipping, re-draw with text labels
!-
SET CLIP "OFF"
SET TRANSFORMATION 1
GRAPH TEXT AT 30,110 : "GOOD FUSE"
DRAW fuse_shell
DRAW Good_fuse
SET TRANSFORMATION 2
GRAPH TEXT AT 30,110 : "BAD FUSE"
DRAW fuse_shell
DRAW Bad_fuse
END
```

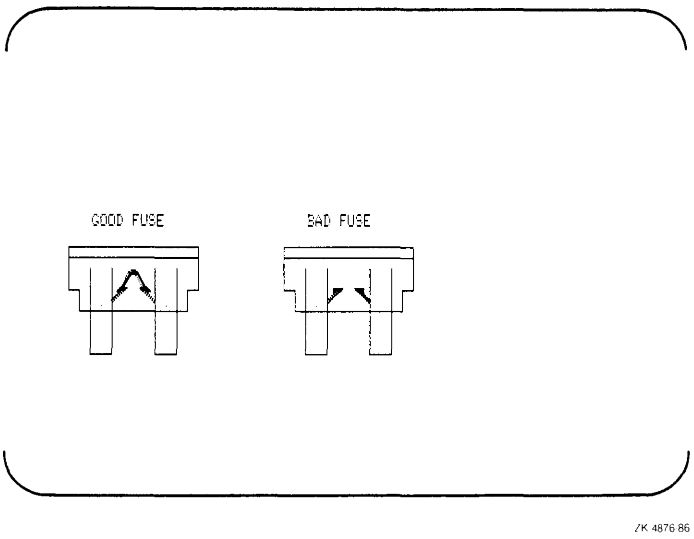
```
PICTURE fuse_shell
DECLARE LONG loop
DIM SINGLE outer_x(8), outer_y(8),                            &
               inner_x(3), inner_y(3),                        &
               edge_x(3), edge_y(3)
   DATA 20,90,80,90,80,70,75,70,                              &
        75,60,25,60,25,70,20,70,20,90
   READ outer_x(loop),outer_y(loop) FOR loop = 0% TO 8%
   DATA 30,80,30,40,40,40,40,80
   READ inner_x(loop),inner_y(loop) FOR loop = 0% TO 3%
   DATA 70,80,70,40,60,40,60,80
   READ edge_x(loop),edge_y(loop) FOR loop = 0% TO 3%
   SET LINE SIZE 1
   SET LINE COLOR 1
   MAT GRAPH LINES outer_x,outer_y
   MAT GRAPH LINES inner_x,inner_y
   MAT GRAPH LINES edge_x,edge_y
   GRAPH LINES 20,85; 80,85
   SET POINT STYLE 1
   SET POINT COLOR 2
   GRAPH POINTS 35,62.5;65,62.5
END PICTURE

PICTURE Good_fuse
   DECLARE LONG loop
   DIM SINGLE fuse_x(10),fuse_y(10)
   DATA 40,65,45,70,43,70,45,72,48,78,50,79,52,                 &
        78,55,72,57,70,55,70,60,65
   READ fuse_x(loop),fuse_y(loop)  FOR loop = 0% TO 10%
   SET LINE SIZE 4
   SET LINE COLOR 2
   MAT GRAPH LINES fuse_x,fuse_y
END PICTURE
```

```
PICTURE Bad_fuse
   DECLARE LONG loop
   DIM SINGLE left_x(2),left_y(2),right_x(2),right_y(2)
   DATA 40,65,57,70,45,70,55,70,43,70,60,65
   READ left_x(loop), left_y(loop),                       &
        right_x(loop), right_y(loop)  FOR loop = 0% TO 2%
   SET LINE SIZE 4
   SET LINE COLOR 2
   MAT GRAPH LINES left_x,left_y
   MAT GRAPH LINES right_x,right_y
END PICTURE
```

Output Screen 1:



Output Screen 2:



A complementary ASK CLIP statement allows you to retrieve the clipping status in a string variable you supply. You can then reset the clipping status with this variable later in your program.

```
ASK CLIP clip_stat
```

mmary

Creating a graphics image can involve each of the following steps:

- Setting world window boundaries with the SET WINDOW statement
- Setting world viewport boundaries with the SET VIEWPORT statement
- Setting the current transformation explicitly with the SET TRANSFORMATION statement, or implicitly with the SET WINDOW or SET VIEWPORT statement
- Clipping the graphics image at the world viewport boundaries with the SET CLIP statement

None of these steps is required. VAX BASIC includes convenient default
so that, if you choose to, you can safely ignore these options. However,
setting the world window boundaries alone provides you with a significa
increase in flexibility for minimum effort. Defining transformations allow
you to create multiple images and complex screens. Another kind of
transformation, known as a device transformation, is possible. Device
transformations are described in Chapter 8.

This chapter has introduced statements that can add a great deal of
flexibility to your programs. Graphics subprograms (pictures) take full
advantage of this additional flexibility. The next chapter shows you how
to use pictures to create complex images.

Chapter 6

Creating Complex Images

When you invoke a graphics subprogram, or *picture*, you can easily perform various transformations and create complex images. This chapter explains how to define VAX BASIC graphics pictures of varying complexity and invoke them with a variety of transformations.

.1 Pictures

A graphics picture is the definition of an image in a block of code. The definition is delimited by the PICTURE and END PICTURE statements. The picture must have a valid VAX BASIC identifier and the name must not be the same as a SUB or FUNCTION subprogram, or a PROGRAM unit.

```
PICTURE which_picture
    .
    .
    .
END PICTURE
```

This block of code can be invoked in a fashion similar to a VAX BASIC SUB or FUNCTION subprogram, and follows the standard VAX/VMS calling procedures for passing parameters. Pictures should be declared with the EXTERNAL statement in the program unit that invokes the picture. You invoke a picture with the DRAW statement.

```
PROGRAM calling
  EXTERNAL PICTURE sample_name(SINGLE)
  DRAW sample_name(val_ue)
    .
    .
    .
END PROGRAM

PICTURE sample_name(SINGLE val_ue)
    .
    .
    .
END PICTURE
```

VAX BASIC provides the END PICTURE and EXIT PICTURE statements to terminate execution of a picture. Execution of a STOP statement withi a picture definition terminates execution of the entire program.

6.1.1 Simple Pictures

You set up the definition of a graphics image in world coordinates within the PICTURE and END PICTURE statements. As with other VAX BASIC subprograms, you can pass parameters to a picture.

The following picture defines a rectangle in terms of the left, right, bottor and top boundaries. These boundaries are used to specify the world coordinates of each corner of the rectangle.

Example

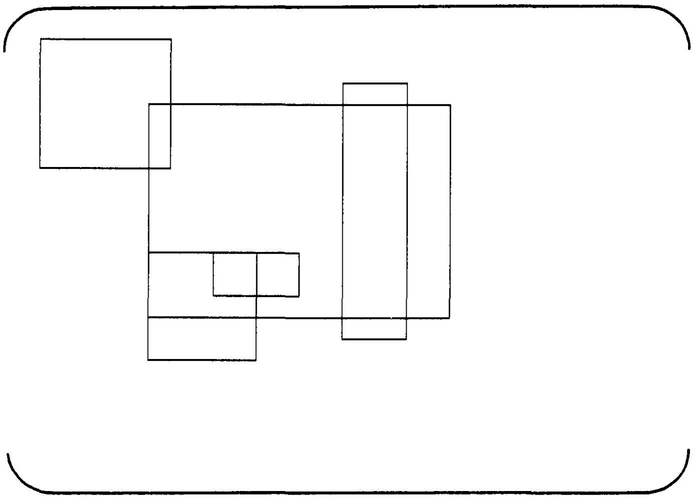
```
PICTURE box(SINGLE left_1,right_1,bottom,top)
   PLOT LINES : left_1,top;         &
                right_1,top;        &
                right_1,bottom;     &
                left_1,bottom;      &
                left_1,top
END PICTURE
```

To invoke this picture, use the DRAW statement at an appropriate point in your program. The DRAW statement passes parameters to the picture. Parameters passed from the DRAW statement to the picture must agree ir number and data type with the parameters listed in the picture definition. The picture *box* requires values for the box boundaries in the order specified. The DRAW statements in the following example invoke the picture *box* to draw several rectangles on the display area. Note that no parameter-passing mechanisms are required in the DRAW or PICTURE statements because pictures are VAX BASIC subprograms and use the default passing mechanisms. After compilation, the main program and any separately compiled programs should be linked together before execution.

Example

```
PROGRAM draw_boxes
  EXTERNAL PICTURE box(SINGLE,SINGLE,SINGLE,SINGLE)
  SET WINDOW 0,100,0,100

  DRAW box(0,30,60,90)
  DRAW box(70,85,20,80)
  DRAW box(25,50,15,40)
  DRAW box(40,60,30,40)
  DRAW box(25,95,25,75)
END PROGRAM
```

Output


ZK-4933-86

The following picture is an example that defines the image of an ellipse. The ellipse requires values for the height and width to be passed with the DRAW statement. This picture is based on the general formula of an ellipse centered at the point 0,0:

$$\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$$

The variable *a* is one half of the major axis dimension of the ellipse, and *b* is one half of the minor axis dimension. This formula yields an ellipse whose major axis is on the y axis; later in the program *a* becomes *Half_height*, and *b* becomes *Half_width*. Solving this equation for *y* yields the alternate form:

$$\frac{y^2}{b^2} = 1 - \frac{x^2}{a^2}$$

or

$$y^2 = b^2 * (1 - \frac{x^2}{a^2})$$

Because a^2 and b^2 are used in the calculation of each point, and because they are constant during the execution of this picture, separate variables are set aside for each, and calculated only once. A^2 is called *Half_height_squared* and b^2 is *Half_width_squared.*

Example

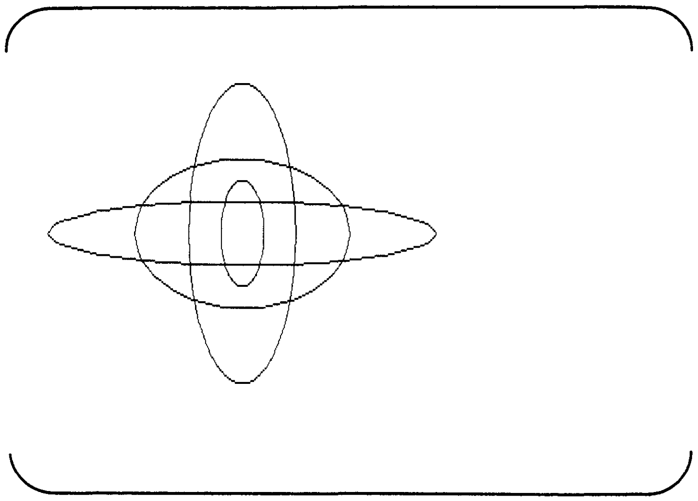
```
%TITLE "ELLIPSE - Draw an ellipse centered at 0,0"
%IDENT "X1.000"
PICTURE ellipse (SINGLE Width, Height_1)
    OPTION TYPE = EXPLICIT
    DECLARE LONG CONSTANT Number_of_steps = 60
    DECLARE SINGLE  Half_width,           &
                    Half_height,          &
                    Half_width_squared,   &
                    Half_height_squared,  &
                    Increment,            &
                    X_pos,Y_pos
    Half_height = Height_1/ 2
    Half_width = Width / 2
    Half_height_squared = Half_height * Half_height
    Half_width_squared = Half_width * Half_width
    Increment = Width/Number_of_steps
    !+
    ! Draw the top of the ellipse
    !-
    PLOT LINES -Half_width, 0;
    FOR X_pos = -Half_width + Increment TO Half_width STEP Increment
        Y_pos = SQRT (ABS (Half_height_squared                        &
                   * (1-X_pos*X_pos/Half_width_squared)))
       PLOT LINES X_pos, Y_pos;
    NEXT X_pos
    !+
    ! Because the FOR loop above may not exactly terminate
    ! at +Half_width due to the imprecision of floating point arithmetic,
    ! complete the top half of the ellipse:
    !-
    PLOT LINES Half_width, 0;
    !+
    ! Draw the bottom of the ellipse
    !-
    FOR X_pos = Half_width - Increment TO -Half_width STEP -Increment
        Y_pos = - SQRT (ABS(Half_height_squared                      &
                   * (1-X_pos*X_pos/Half_width_squared)))
        PLOT LINES X_pos, Y_pos;
    NEXT X_pos
    !+
    ! Because the FOR loop above may not exactly terminate
    ! at -Half_width because of the imprecision of floating point arithmetic,
    ! complete the bottom half of the ellipse:
    !-
    PLOT LINES -Half_width, 0
END PICTURE
```

You must supply values for the width and height of the ellipse with the DRAW statement. The following example invokes the picture *ellipse* four times.

Example

```
PROGRAM call_ellipses
  EXTERNAL PICTURE ellipse(SINGLE,SINGLE)
  SET WINDOW -0.5,0.5,-0.5,0.5
  DRAW ellipse(0.1,0.25)
  DRAW ellipse(0.25,0.7)
  DRAW ellipse(0.5,0.35)
  DRAW ellipse(0.9,0.15)
END PROGRAM
```

Output


ZK-4931-86

Window and viewport boundaries are set in the main program before pictures are invoked. You cannot set the boundaries of windows and viewports or alter the clipping status within a picture definition. The following statements are invalid when used within a picture definition:

- SET WINDOW
- SET VIEWPORT
- SET CLIP
- SET TRANSFORMATION
- SET DEVICE WINDOW

- SET DEVICE VIEWPORT
- SET INPUT PRIORITY

.1.2 Pictures Within Pictures

A picture definition can contain DRAW statements to invoke other pictures, as shown in the following example. Note that the picture *room* is declared with the EXTERNAL statement in the picture *house*.

Example

```
PICTURE house
  EXTERNAL PICTURE room
    .
    .
    .
  DRAW room
END PICTURE

PICTURE room
    .
    .
    .
END PICTURE
```

The following picture, *face*, declares three other pictures with the EXTERNAL statement, then invokes each of the pictures in turn.

Example

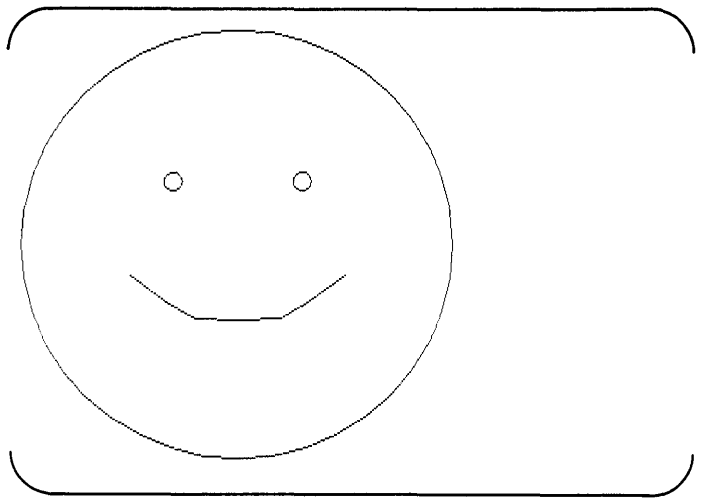
```
PICTURE face
    EXTERNAL PICTURE head, eyes, mouth
    DRAW head
    DRAW eyes
    DRAW mouth
END PICTURE

PICTURE head
    DECLARE LONG CONSTANT npoints = 40
    DECLARE SINGLE turn,increment,radius,            &
            LONG loop_count
    DIM SINGLE xs(40), ys(40)
    increment = 2 * PI/npoints
    turn = 0
    radius = 0.5
    FOR loop_count = 0% TO (npoints - 1%)
      xs(loop_count) = COS (turn) * radius + 0.5
      ys(loop_count) = SIN (turn) * radius + 0.5
      turn = turn + increment
    NEXT loop_count
    xs(npoints) = xs(0)
    ys(npoints) = ys(0)
    MAT PLOT LINES xs,ys
END PICTURE

PICTURE EYES
    SET POINT STYLE 4
    PLOT POINTS 0.35,0.65; 0.65,0.65
END PICTURE

PICTURE MOUTH
  DECLARE LONG counter
  DIM SINGLE mouth_x(6),mouth_y(6)
  DATA 0.25,0.34,0.4,0.5,0.6,0.66,0.75,              &
       0.425,0.36,0.325,0.324,0.325,0.365,0.425
  READ mouth_x(counter) FOR counter = 0 TO 6
  READ mouth_y(counter) FOR counter = 0 TO 6
  MAT PLOT LINES mouth_x,mouth_y
END PICTURE
```

When invoked with the DRAW statement, this picture creates one face:

Example

```
EXTERNAL PICTURE face
SET WINDOW 0,1,0,1
DRAW face
END
```

Output


ZK-4928-86

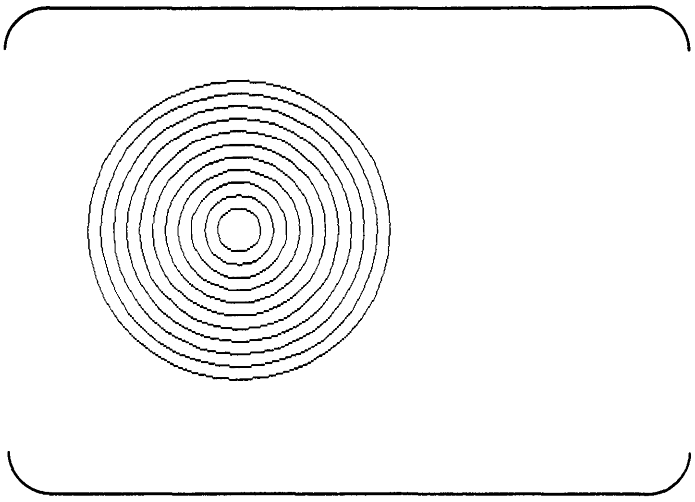
1.3 Recursive Pictures

A picture definition can be recursive; that is, a picture definition can include a DRAW statement that invokes the same picture again. For recursive pictures, the picture must be declared with the EXTERNAL statement within the picture definition as well as in the program unit that first calls the picture. The following picture defines a tunnel made up of concentric circles. The picture invokes itself with a smaller radius each time. After the requested number of circles is drawn, the EXIT PICTURE statement terminates execution of the picture.

Example

```
OPTION ANGLE = DEGREES
EXTERNAL PICTURE tunnel(SINGLE , ,LONG)
SET WINDOW -0.5,0.5,-0.5,0.5
DRAW tunnel(0.05,0.03,11)
END

PICTURE tunnel(SINGLE radius,variation,LONG counter)
  OPTION TYPE = EXPLICIT
  DECLARE LONG loop_count,          &
          SINGLE turn,increment,x,y
  DECLARE LONG CONSTANT npoints = 40
  DIM SINGLE xs(npoints), ys(npoints)
  EXTERNAL PICTURE tunnel(SINGLE , ,LONG)

  increment = 2*PI/npoints
  turn = 0
  FOR loop_count = 0% TO npoints - 1%
     xs(loop_count) = COS (turn) * radius
     ys(loop_count) = SIN (turn) * radius
     turn = turn + increment
 NEXT loop_count
 xs(npoints) = xs(0)
 ys(npoints) = ys(0)
 MAT PLOT LINES xs,ys

 IF counter > 1
    THEN DRAW tunnel(radius + variation,variation,(counter-1))
 END IF
END PICTURE
```

Output


ZK-4926-86

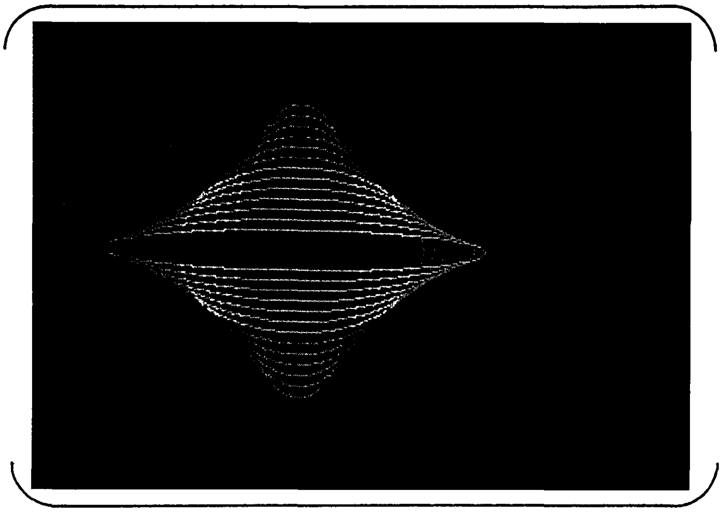
The following example adds code to make the picture *ellipse* from Section 6.1.1 recursive. The picture creates 18 ellipses. With each invocation, the value of *Height_1* is increased and the value of *width* is decreased.

Example

```
PROGRAM pattern_ellipses
  EXTERNAL PICTURE recursive_ellipse(SINGLE,SINGLE,LONG)
  SET WINDOW -0.5,0.5,-0.5,0.5
  DRAW recursive_ellipse(0.9,0.1,0)
END PROGRAM
```

```
PICTURE recursive_ellipse(SINGLE width, Height_1,LONG counter)
    OPTION TYPE = EXPLICIT
    DECLARE LONG CONSTANT Number_of_steps = 60
    DECLARE SINGLE  Half_width,          &
                    Half_height,         &
                    Half_width_squared,  &
                    Half_height_squared, &
                    Increment,           &
                    X_pos,Y_pos
    Half_height = Height_1/ 2
    Half_width = width / 2
    Half_height_squared = Half_height * Half_height
    Half_width_squared = Half_width * Half_width
    Increment = Width/Number_of_steps
    PLOT LINES -Half_width, 0;
    FOR X_pos = -Half_width + Increment TO Half_width STEP Increment
        Y_pos = SQRT (ABS (Half_height_squared *                     &
                     (1-X_pos*X_pos/Half_width_squared)))
       PLOT LINES X_pos, Y_pos;
    NEXT X_pos
    PLOT LINES Half_width, 0;

    FOR X_pos = Half_width - Increment TO -Half_width STEP -Increment
        Y_pos = -SQRT (ABS (Half_height_squared *                   &
                      (1-X_pos*X_pos/Half_width_squared)))
        PLOT LINES X_pos, Y_pos;
    NEXT X_pos
    PLOT LINES -Half_width, 0
    !+
    !Additional code for recursion
    !-
    IF counter < 17
       THEN
       width = width - 0.05
       Height_1 = Height_1 + 0.05
       IF counter > 12
          THEN SET LINE COLOR 3
          ELSE IF counter > 6
                  THEN SET LINE COLOR 2
                  END IF
          END IF
      DRAW recursive_ellipse(width,Height_1,(counter + 1))
      ELSE
      EXIT PICTURE
   END IF
END PICTURE
```

Output



.2 Invoking Pictures with Transformation Functions

The examples so far have used a simple form of the DRAW statement to invoke a picture. The DRAW statement can also include optional transformation functions. These optional functions allow you to control the positioning of the picture.

To change an image in size, shape, or position, you usually define transformations as explained in Chapter 5. For instance, you can change the position of an image by altering the window or viewport settings. However, when the image is defined in a picture, you can transform the image with VAX BASIC built-in functions. When you include a transformation function in the DRAW statement, the picture is transformed during execution. The built-in transformation functions include:

- SHIFT
- SCALE
- ROTATE

- SHEAR
- TRANSFORM

In addition, you can define your own transformation functions to use witl the DRAW statement. See Section 6.3 for more information.

Each transformation function has a unique effect on the points plotted by output statements within the picture definition. These transformation functions affect points only in PLOT and MAT PLOT statements; points in GRAPH and MAT GRAPH statements within a picture definition are not affected by the transformation functions included with the DRAW statement. Therefore, when you want output statements to be affected by the transformation functions, use PLOT and MAT PLOT statements in subprograms. See Chapter 7 for information about accepting input durin execution of a picture.

When you know that transformation functions will not be used, it is mor efficient to use GRAPH statements rather than PLOT statements inside picture definitions.

Note that when text is drawn within a picture, text attributes, such as the starting point and the height, are not affected by the transformation functions in a DRAW statement; for more information on transforming te output, see the TRANSFORM function in Chapter 9.

The following sections describe the effects of each of these transformatio functions.

6.2.1 Shifting Pictures

You can move or shift pictures by including the transformation function SHIFT with the DRAW statement. The amounts of shifting you want to apply to each of the x- and y-coordinates in an image are supplied as arguments to the SHIFT function. The x- and y-coordinates of the points are moved or shifted by the amounts specified, as in this example:

```
DRAW boxes WITH SHIFT(move_x,move_y)
```

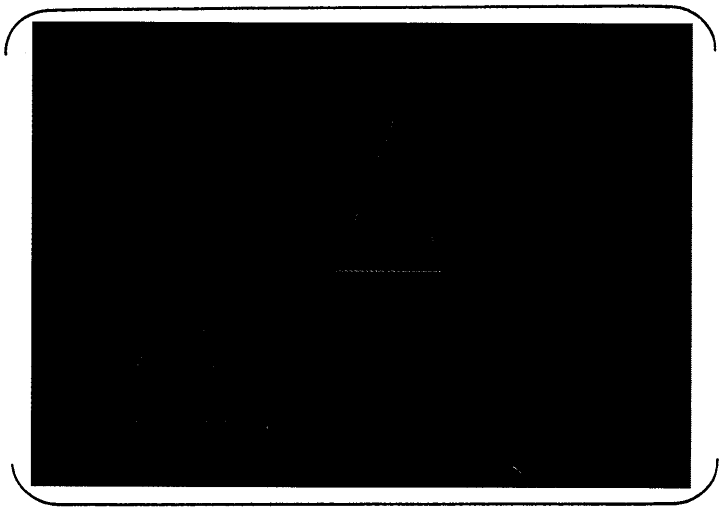
The value of *move_x* is added to each x-coordinate for output statements specified in the picture so that the new value of the x-coordinate is *x* + *move_x*. The value of *move_y* is added to each y-coordinate so that the resulting value for the y-coordinate is *y* + *move_y*.

In the following example, the image of a triangle is first drawn in red without any transformation function applied. The second invocation shifts the triangle to the right and is displayed in blue. The value of the y-coordinates is not altered in the blue triangle because the value for *move_y* is zero. In the third DRAW statement, the color is changed to green. Both the x- and y-coordinates are changed and the triangle is shifted to the right and moved up. The program uses the default world viewport, but sets the world window to 0,100,0,100 before the picture is invoked.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE triangle
DECLARE LONG CONSTANT red = 2,blue = 3,green = 1
DECLARE SINGLE move_x,move_y

SET WINDOW 0,100,0,100
SET LINE COLOR red
DRAW triangle
move_x = 30
move_y = 0
SET LINE COLOR blue
DRAW triangle WITH SHIFT(move_x,move_y)
move_x = 50
move_y = 35
SET LINE COLOR green
DRAW triangle WITH SHIFT(move_x,move_y)
END

PICTURE triangle
  PLOT LINES 10,10; 25,50; 35,10; 10,10
END PICTURE
```

Output

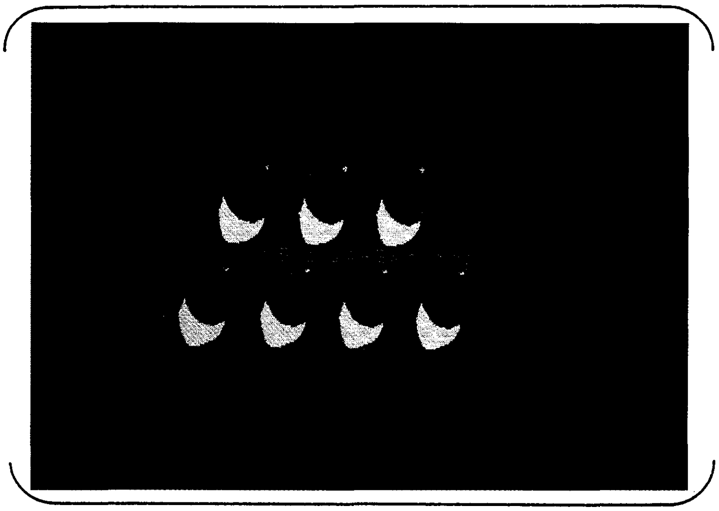


In the following example, the image of a swan is defined in a picture. Th picture is invoked seven times with the SHIFT function. Unlike the pictuı *swan* in Chapter 5, this picture uses PLOT statements rather than GRAPI statements. The window is set to -200,200,-200,200 before the picture is invoked.

Example

```
PICTURE swan
 OPTION TYPE = EXPLICIT
 DECLARE LONG counter
 DIM SINGLE x_array(98),y_array(98),                              &
            mark_x(3),mark_y(3),                                  &
            eye_x(2),eye_y(2),                                    &
            wing_area_x(28),wing_area_y(28)
 Outline_data:
  DATA 82,80.5,80,80.5,75,81,72,81.5,70,83.5,68,84,65,84.5,       &
  63,83,61,82,60.5,80,60.55,78,62.5,75,64,70,66.5,67,70,62.5,     &
  72,60,75,57,78,54,80,50,82.5,46,84,40,86.5,30,87,24.5,87,20,    &
  86,17.5,83.5,13.5,82,12,80,11.5,77,10,70,9,65,9,60,9.2,53,10,   &
  50,10.5,40,12,35,14,30,18,27,20,20,27.5,14,36,12,40,8.5,50,8,   &
  59,10,52.5,14,48,20,45,17,50,16.75,57,20,60,18,61.5,20,67,27,   &
  75,27.5,72,37,85,36,80,34,75,32.25,70,32,65,32.25,60,32.35,     &
  58,34,53,35,50,38,45,40,41,43.5,38,48,34,52,32.5,60,31,63,32,   &
  69.5,35.5,70.5,37,72,40,70.5,43,69.5,47,68.5,50,65.5,55,62.5,   &
  60,60,65,57.5,70,55,75,54,80,54.25,83,54,84.5,55,87,59,90,60,   &
  91,63,92.25,66.5,93,68,92.75,70,92,71.5,91,73,89.5,74,87,75.5,  &
  85,78,84,80,82,82,82,80,80.5,82,80.5

 Wing_area_data:
  DATA 32.35,58,34,53,35,50,38,45,40,41,43.5,38,48,34,52,32.5,    &
  60,31,63,32,69.5,35.5,70.5,37,32.35,58,30.05,50,28,42,27.5,     &
  44.5,28.5,30,30,24,31.5,20,35,14,40,12,45,13,50,14,53,15,       &
  58.5,17.5,61,19,64.5,22,69,30,70.5,37

 Head_mark_data:
  DATA 73,87,70,86,73,85,73.15,86,67,89,69,88,70.5,90

 READ x_array(counter),y_array(counter) FOR counter = 0% TO 98%
 READ wing_area_x(counter),wing_area_y(counter) FOR counter = 0% TO 28%
 READ mark_x(counter),mark_y(counter) FOR counter = 0% TO 3%
 READ eye_x(counter), eye_y(counter) FOR counter = 0% TO 2%

 Display_the_swan:
  SET LINE COLOR 3%
  SET LINE STYLE 1
  MAT PLOT LINES x_array,y_array
  SET AREA STYLE "PATTERN"
  MAT PLOT AREA : wing_area_x,wing_area_y
  MAT PLOT AREA mark_x,mark_y
  SET LINE COLOR 2%
  MAT PLOT LINES eye_x,eye_y
END PICTURE

PROGRAM swimming_swans
  EXTERNAL PICTURE swan
  SET WINDOW , TRAN 1 : -200,200,-200,200

  DRAW swan WITH SHIFT(50,0)
  DRAW swan WITH SHIFT(-25,0)
  DRAW swan WITH SHIFT(-100,0)
  DRAW swan WITH SHIFT(-135,-100)
  DRAW swan WITH SHIFT(-60,-100)
  DRAW swan WITH SHIFT(15,-100)
  DRAW swan WITH SHIFT(90,-100)
  !+
  !Adjust the text height for the new window
  !-
  SET TEXT HEIGHT 10%
  GRAPH TEXT AT -50,-5 : "Seven Swans A-Swimming"
END PROGRAM
```

Output



If the picture *swan* had used MAT GRAPH statements instead of MAT PLOT statements, the image of the swan would not have been shifted at all. Instead, each swan would have been displayed in exactly the same position as in the first invocation.

2.2 Scaling Pictures

The SCALE function allows you to increase or decrease the size of the image produced by a picture. You can scale the width and the height of an image by a scale factor you supply. The following statement supplies arguments for both the height and the width with SCALE.

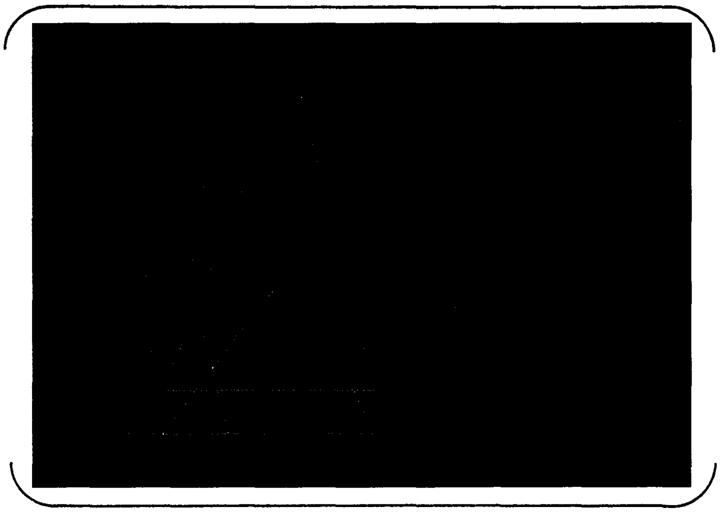
```
DRAW picture_name WITH SCALE(2,3)
```

This DRAW statement multiplies each x-coordinate in the PLOT statement of the picture by 2. Each y-coordinate is multiplied by 3. The effect of this statement and other DRAW WITH SCALE statements are illustrated in the following example. The picture *triangle* is invoked at first with no transformation function. The second DRAW statement scales only the x-coordinates (displayed in red). The third invocation of the triangle scales only the y-coordinates (displayed in blue). The final invocation of the triangle scales both the x- and y-coordinates. Notice that the triangle is not shifted. An example in Section 6.2.5 demonstrates how to combine the SHIFT and SCALE functions.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE triangle
DECLARE LONG CONSTANT red = 2,              &
             blue = 3, green = 1
SET WINDOW 0,100,0,100
DRAW triangle
SET LINE COLOR red
DRAW triangle WITH SCALE(2,1)
SET LINE COLOR blue
DRAW triangle WITH SCALE(1,2)
SET LINE COLOR green
DRAW triangle WITH SCALE(2)
END
```

Output

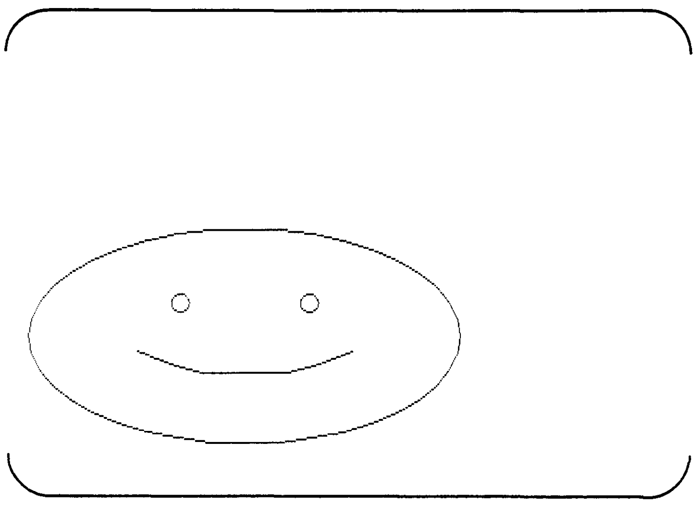


The following example illustrates the effect of reducing the y-coordinates while leaving the x-coordinates unchanged. The DRAW statement invok the picture *face,* multiplies each x-coordinate by 1 (leaving it unchanged) and divides each y-coordinate by 2.

Example

```
EXTERNAL PICTURE face
SET WINDOW 0,1,0,1
DRAW face WITH SCALE(1,0.5)
END
```

Output



6.2.3 Rotating Pictures

Images defined in picture definitions can be rotated in a counterclockwise direction about the point of origin (point 0,0). You specify the amount of rotation as an argument to the ROTATE function. The amount of rotation can be in radians or degrees, according to the OPTION ANGLE statement for the program unit. Negative values cause rotation in a clockwise direction. The following DRAW statement invokes a picture with a rotation of 15° in a counterclockwise direction.

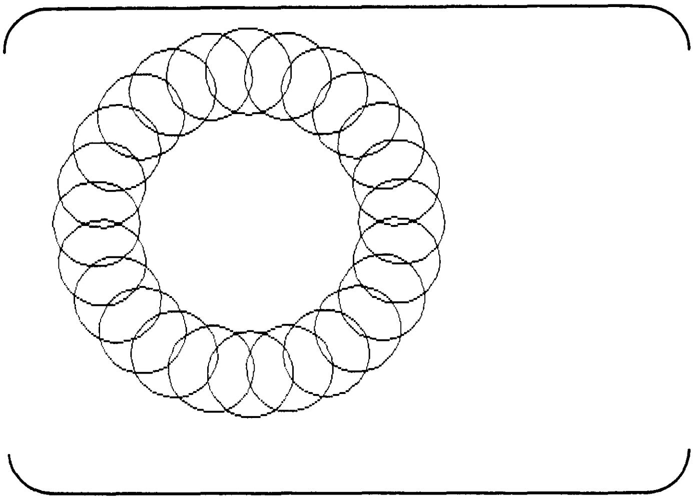
```
OPTION ANGLE = DEGREES
DRAW picture_name WITH ROTATE(15)
```

The following program draws the picture *circle* 24 times with an increasing value for the rotation. The world window is set to place the point 0,0 in the center of the window. The picture is rotated about the center point.

Example

```
PROGRAM call_circles
 OPTION ANGLE = DEGREES
 OPTION TYPE = EXPLICIT
 DECLARE LONG loop
 EXTERNAL PICTURE circle(SINGLE)
 SET WINDOW -1,1,-1,1

 FOR loop = 1 TO 24
   !+
   !Draw circle with radius 0.2
   !-
   DRAW circle(0.2) WITH ROTATE(15 * loop)
 NEXT loop
END PROGRAM

PICTURE circle(SINGLE radius)
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE angle
  DECLARE LONG CONSTANT npoints = 40
  DECLARE SINGLE CONSTANT increment = 2*PI/npoints
  DECLARE LONG loop_count
  DIM SINGLE xs(40), ys(40)

  angle = 0
  FOR loop_count = 0% TO npoints - 1%
     xs(loop_count) = COS (angle) * radius + 0.5
     ys(loop_count) = SIN (angle) * radius + 0.5
     angle = angle + increment
  NEXT loop_count
  xs(npoints) = xs(0)
  ys(npoints) = ys(0)
  MAT PLOT LINES xs,ys
END PICTURE
```

Output


ZK-4929-86

The point of origin need not be within the world window. For example, an image in the window 25,50,25,50 can be rotated about the point 0,0 even though this point is not in the window.

Note that pictures can be rotated about any point you choose. The following statement rotates the picture *tryit* about the point x,y.

```
DRAW tryit WITH SHIFT(-x,-y) * ROTATE(angle) * SHIFT(x,y)
```

The examples in this section rotate pictures about the point of origin. See the DRAW statement in Chapter 9 for an example of rotating a picture about other points.

6.2.4 Shearing Pictures

The SHEAR function allows you to skew the coordinates of an image so that the image appears to tilt or lean over. The exact placement depends on the argument you supply with SHEAR and the relative position of the point of origin (0,0). You can supply the argument in either degrees or radians, depending on the option you specify with the OPTION ANGLE statement. The following statement changes the coordinates of a point x,y to the coordinates (x + y * TAN(25)),y.

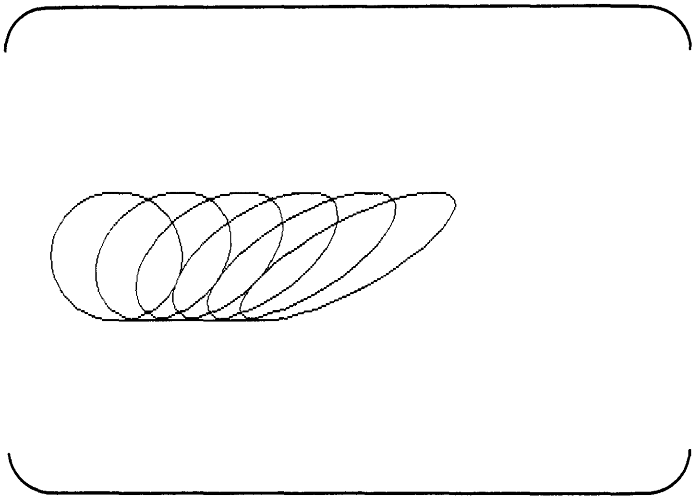
```
OPTION ANGLE = DEGREES
DRAW picture_name WITH SHEAR(25)
```

The following example uses SHEAR to skew the picture *ball* (a circle) several times to illustrate the effects of various arguments. Notice that unlike the ROTATE function, SHEAR does not change the values of the y-coordinates.

Example

```
PROGRAM Spring
 OPTION ANGLE = DEGREES
 EXTERNAL PICTURE coil(SINGLE)
 DECLARE LONG loop_count

 FOR loop_count = 0% to 5%
   !Draw coil with radius of 0.15
   DRAW coil(0.15) WITH SHEAR(10 * loop_count)
 NEXT loop_count
END PROGRAM
```

Output



.2.5 Combining Transformations

You can invoke a picture with more than one transformation function. To use more than one transformation function with the DRAW statement, separate the functions with an asterisk (*). For instance, the following DRAW statements each use a combination of two transformation functions:

Example

```
DRAW face WITH SHEAR(30) * SCALE(1,2)
DRAW face WITH SHIFT(3,0.5) * SCALE (2)
```

Transformation functions are not associative; therefore, the order of the functions on the DRAW statement is important when you combine transformations. For instance, SHIFT * SCALE does not produce the same new coordinates as SCALE * SHIFT. This is comparable to the importance of the order of operations in a numeric expression.

In Section 6.2.2, the scaled triangles are displayed without any shifting, making the individual triangles difficult to differentiate. The following example draws the picture *triangle* by combining the SHIFT and SCALE functions.

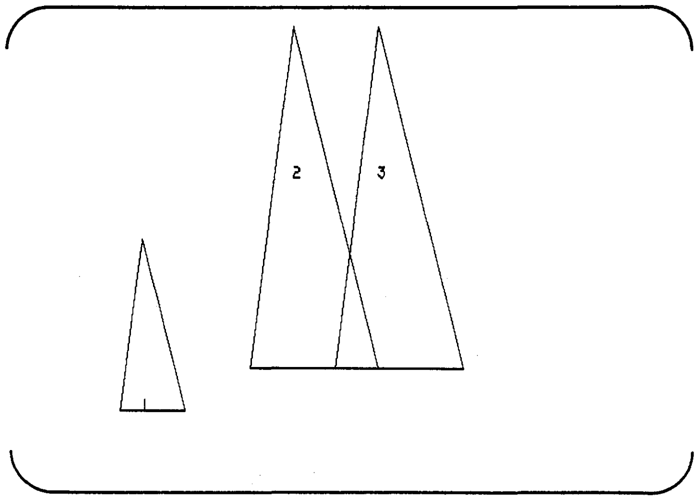
In the following example, the picture *triangle* is invoked three times with the DRAW statement as follows:

1. With no transformation
2. With SCALE(2) * SHIFT(20,0)
3. With SHIFT(20,0) * SCALE(2)

Example

```
PROGRAM draw_triangles
 EXTERNAL PICTURE triangle
 SET WINDOW, TRAN 1 : 0,100,0,100
 SET TEXT HEIGHT 3
 !+
 !Triangle 1 in output
 !-
 DRAW triangle
 GRAPH TEXT AT 15,10 : "1"
 !+
 !Triangle 2 in output
 !-
 DRAW triangle WITH SCALE(2) * SHIFT(20,0)
 GRAPH TEXT AT 50,65 : "2"
 !+
 !Triangle 3 in output
 !-
 DRAW triangle WITH SHIFT(20,0) * SCALE(2)
 GRAPH TEXT AT 70,65 : "3"
END PROGRAM
```

Output



ZK-5514-86

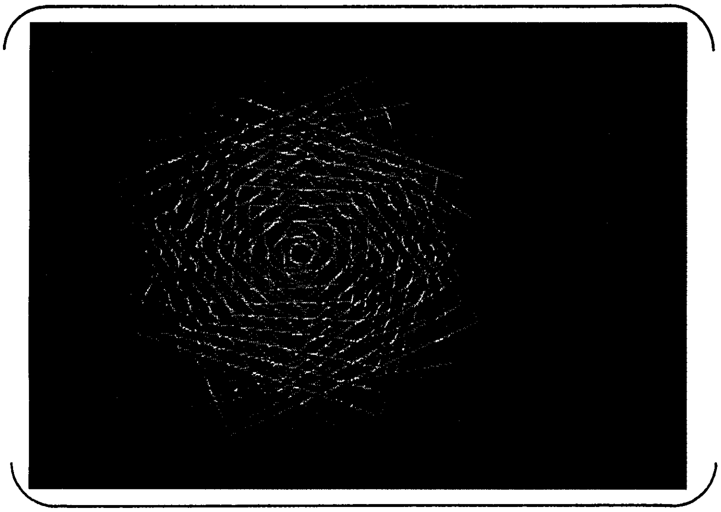
The following example combines the ROTATE and SCALE transformation functions. Each invocation of the picture *square* increases the size of the square and turns the square by an additional 0.25 radians. The window is set to -1,1,-1,1 so that the point of origin is in the center of the screen.

Example

```
PROGRAM call_square
  OPTION TYPE = EXPLICIT
  OPTION ANGLE = RADIANS
  EXTERNAL PICTURE square
  DECLARE SINGLE counter
  SET WINDOW -1,1,-1,1

  FOR counter = 1 TO 10 STEP 0.25
    IF color_var < 3
       THEN color_var = color_var + 1
       ELSE color_var = 1
    END IF
    DRAW square WITH ROTATE(counter) * SCALE(counter,counter)
  NEXT counter
END PROGRAM
```

```
PICTURE square
  PLOT LINES : -0.05, 0.05;         &
                0.05, 0.05;         &
                0.05,-0.05;         &
               -0.05,-0.05;         &
               -0.05,0.05
END PICTURE
```

Output



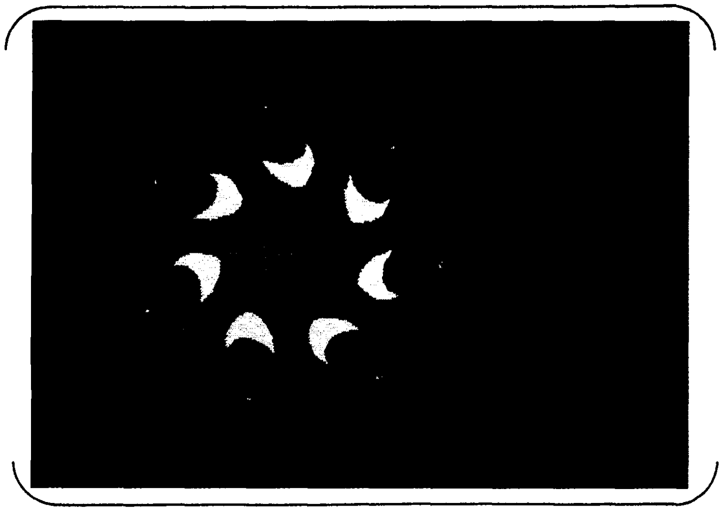
The following example shifts and rotates the picture *swan* counterclockwise about the center point of the world window. Each invocation of the picture rotates the image of the swan 51° from the previous invocation. The point of origin is at the center of the screen.

Example

```
OPTION TYPE = EXPLICIT
OPTION ANGLE = DEGREES
EXTERNAL PICTURE swan
DECLARE SINGLE CONSTANT turn = 51
DECLARE LONG loop

SET WINDOW , TRAN 1 : -200,200,-200,200

FOR loop = 1 TO 7
  DRAW swan WITH SHIFT(21,21) * ROTATE(turn * loop)
NEXT loop
```

```
SET TEXT HEIGHT 12%
GRAPH TEXT AT -40,0 : "Seven Swans"
GRAPH TEXT AT -38,-12 : "A-Spinning"
END
```

Output



Nested picture invocations can also have transformation functions. Such nested picture definitions can lead to an accumulation of transformations. There is no limit to the number of levels you can nest pictures; however, nesting several levels deep makes your program extremely difficult to read or debug.

When picture invocations are nested within other pictures, the original transformation function applies throughout the execution of the picture in addition to any other transformation function used. For instance, the following picture definition includes invocations using transformation functions as well as a recursive picture invocation.

Example

```
PICTURE outside
  EXTERNAL PICTURE inside1, inside2, outside
  DECLARE SINGLE angle_var,move_x,move_y
  DECLARE LONG counter

  counter = 1
  DRAW inside1
  DRAW inside2 WITH SHEAR(angle_var)
  IF counter > 10
     THEN DRAW outside WITH SHIFT(move_x,move_y)
  END IF
  counter = counter  + 1
END PICTURE

PROGRAM calling
  EXTERNAL PICTURE outside
  DRAW outside WITH SHIFT(1,0)
END PROGRAM
```

When the picture *outside* is invoked with the transformation function SHIFT, picture *inside1* and *inside2* are invoked with an implicit SHIFT function. Picture *inside2* includes a SHEAR function, which is applied in addition to the invocation with SHIFT.

The recursive call of picture *outside* also includes a transformation function. The original invocation with the transformation function SHIFT is still applied in addition to the SHIFT function in the recursive call. Each recursive invocation builds on the accumulated transformations up to tha point.

The following section discusses how to define your own transformation matrices. If you are content with the transformation functions VAX BASI provides, then move on now to Chapter 7, which discusses different way of supplying graphics input to your programs.

6.3 Defining Your Own Transformation Matrices

When you use a transformation function, each point plotted in the pictur is transformed into a point with new coordinates. VAX BASIC translates each point by multiplying it by a 4 by 4 matrix. The transformation functions described in this chapter each represent a particular 4 by 4 matrix for the transformation of a point. The following chart displays the matrix represented by each VAX BASIC transformation function. Note that these matrices actually have 5 rows and 5 columns; the elements in row and column zero are used by VAX BASIC and are not shown here. The matrices have lower bounds of zero and upper bounds of 4.

Figure 6–1: Matrices for VAX BASIC Transformation Functions

SHIFT(A,B)

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ A & B & 0 & 1 \end{matrix}$$

SCALE(A,B)

$$\begin{matrix} A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

SCALE(A)

$$\begin{matrix} A & 0 & 0 & 0 \\ 0 & A & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

ROTATE(A)

$$\begin{matrix} \text{COS(A)} & \text{SIN(A)} & 0 & 0 \\ \text{-SIN(A)} & \text{COS(A)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

SHEAR(A)

$$\begin{matrix} 1 & 0 & 0 & 0 \\ \text{TAN(A)} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

ZK-5354-86

You can supply your own matrix with the DRAW statement in place of any of the VAX BASIC transformation functions. To do this, first, you can declare a two-dimensional array and initialize the elements to be zero. The matrix must be zero-based and have upper bounds of 4 in both directions. In addition, matrices cannot be record arrays. Next, assign the values you want in the appropriate matrix elements. You should not use elements in row and column zero because VAX BASIC uses these elements during the multiplication of these matrices. The following code sets up a matrix that is equivalent to a transformation of SHIFT(A,B):

```
DECLARE SINGLE A,B,                 &
        LONG row,col
DIM SINGLE shift_matrix(4,4)
FOR row = 1 TO 4
    FOR col = 1 TO 4
    shift_matrix(row,col) = 0
    NEXT col
NEXT row

shift_matrix(1,1) = 1
shift_matrix(2,2) = 1
shift_matrix(3,3) = 1
shift_matrix(4,4) = 1
shift_matrix(4,1) = A
shift_matrix(4,2) = B
    .
    .
    .
```

The following two statements are now equivalent:

```
DRAW picture_name WITH shift_matrix
DRAW picture_name WITH SHIFT(A,B)
```

You can define your own matrices and thereby create your own transformation functions. For instance, for frequently used combinations of transformations, you can define a new 4 by 4 matrix and use it in the DRAW statement as an alternative to combining transformation functions. To do this, use the MAT statement. The following example defines a new matrix that combines the transformation functions SCALE(A) and ROTATE(B):

Example

```
OPTION TYPE = EXPLICIT
OPTION ANGLE = DEGREES
EXTERNAL PICTURE kite
DECLARE SINGLE A,B
DIM SINGLE new_matrix(4,4)
MAT new_matrix = SCALE(A) * ROTATE(B)
DRAW kite WITH new_matrix
END
```

Applying this matrix with the DRAW statement scales a set of x- and y-coordinates by the value of *A*, and it also rotates the set of points by *B* degrees.

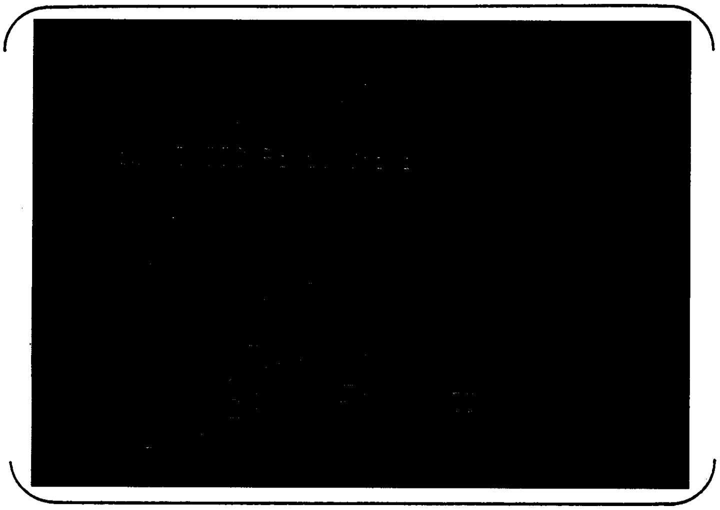
The following example invokes the picture *face* with a variety of transformation functions. The MAT statement is used to create new matrices built on the previous position of the face.

Example

```
!Invoking program
PROGRAM party_crowd
  OPTION TYPE = EXPLICIT
  OPTION ANGLE = DEGREES
  EXTERNAL PICTURE face
  DIM SINGLE next_face(4,4),temp_face(4,4)
  SET WINDOW 0,5,0,5
  !+
  !Draw the party crowd
  !-
  SET LINE COLOR 3
  DRAW face WITH SCALE(1.25,3.25)
  SET LINE COLOR 2
  DRAW face WITH SHIFT(1.5,1.25)
  MAT next_face = SHIFT(1.5,1.25) * SHEAR(15)
  DRAW face WITH next_face
  MAT temp_face = next_face * SHIFT(2,1)
  MAT next_face = temp_face * SHEAR(-30)
  SET LINE COLOR 3
  DRAW face WITH next_face
  MAT temp_face = next_face * SHIFT(-0.5,1.5)
  MAT next_face = temp_face
  DRAW face WITH next_face
  MAT temp_face = next_face * SHEAR(30)
  MAT next_face = temp_face * SCALE(0.75,0.75) * SHIFT(-0.5,1)
  DRAW face WITH next_face
  MAT temp_face = next_face
  MAT next_face = temp_face * SHIFT(0,-3) * SCALE(1,0.75)
  SET LINE COLOR 2
  DRAW face WITH next_face
  MAT temp_face = next_face
  MAT next_face = temp_face * SHIFT(-2,2)
  DRAW face WITH next_face
  MAT temp_face = next_face
  MAT next_face = temp_face * SHIFT(1,1)  * SCALE(0.75,0.75)
  DRAW face WITH next_face
  MAT temp_face = next_face * SHEAR(25)
  MAT next_face = temp_face * SHIFT(1.5,-2.5) * SCALE(1,1.25)
  SET LINE COLOR 3
  DRAW face WITH next_face
  !+
  !No cumulative matrix used for this one
  !-
  SET LINE COLOR 2
  DRAW face WITH ROTATE(300) * SHEAR(60)                    &
              * SHIFT(1.25,0.75)

  SET TEXT HEIGHT 0.25
  GRAPH TEXT AT 0.25,3.4 : "Your BASIC Party Crowd"
END PROGRAM

PICTURE face
  OPTION ANGLE = DEGREES
  EXTERNAL PICTURE head,mouth,eyes
  DRAW head
  DRAW eyes
  DRAW mouth
END PICTURE
```

```
PICTURE head
  OPTION TYPE = EXPLICIT
  DECLARE LONG CONSTANT npoints = 40
  DECLARE SINGLE CONSTANT increment = 2*PI/npoints,     &
                      radius = 0.5
  DECLARE SINGLE angle,                                 &
          LONG loop
  DIM SINGLE xs(40), ys(40)

  angle = 0
  FOR loop_count = 0% TO npoints - 1%
     x = COS(angle) * radius + 0.5
     y = SIN(angle) * radius + 0.5
     xs(loop_count) = x
     ys(loop_count) = y
     angle = angle + increment
  NEXT loop_count

  xs(npoints) = xs(0)
  ys(npoints) = ys(0)
  MAT PLOT LINES xs,ys
END PICTURE

PICTURE mouth
  DECLARE SINGLE mouth_x(6),mouth_y(6),               &
          LONG loop
  DATA 0.25,0.425,0.34,0.36,0.4,0.325,                &
       0.5,0.324,0.6,0.325,0.66,0.365,                &
       0.75,0.425
  READ mouth_x(loop),mouth_y(loop)  FOR loop = 0% TO 6%
  MAT PLOT LINES mouth_x,mouth_y
END PICTURE

PICTURE eyes
    SET POINT STYLE 4
    PLOT POINTS 0.35,0.65; 0.65,0.65
END PICTURE
```

Output



VAX BASIC provides an additional transformation function, TRANSFORM. When used within nested picture invocations, the TRANSFORM function returns the cumulative matrix for all current transformations. Like the other transformation functions, TRANSFORM can be used on the right-hand side of a MAT statement or as a transformation function in the DRAW statement.

A picture drawn from within a picture automatically inherits the transformations for the outer picture as well as the transformations specified in the current invocation. Therefore, using the TRANSFORM function in a DRAW statement causes the picture to be invoked with *twice* the current transformation.

In the following example, the picture *food* is invoked with a transformation of SCALE(2) * SCALE(2).

Example

```
PROGRAM call_tray
  EXTERNAL PICTURE tray
  DRAW tray WITH SCALE(2)
END PROGRAM

PICTURE tray
 OPTION ANGLE = DEGREES
 EXTERNAL PICTURE food
 DRAW food WITH TRANSFORM
END PICTURE
```

In the following example, the picture *three* is invoked with the DRAW statement using the TRANSFORM function. At the time of invocation, TRANSFORM contains values equivalent to ROTATE(45) * SHIFT(1,0); therefore, *three* will be invoked with the equivalent of ROTATE(45) * SHIFT(1,0) * ROTATE(45) * SHIFT(1,0).

Example

```
OPTION ANGLE = DEGREES
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE one, two, three
DRAW one WITH SHIFT(1,0)
END

PICTURE one
  EXTERNAL two
  DRAW two WITH ROTATE(45)
END PICTURE

PICTURE two
  EXTERNAL three
  DRAW three WITH TRANSFORM
END PICTURE
```

You can also use the TRANSFORM function to define your own matrices for the DRAW statement. For more examples, see the DRAW statement and the TRANSFORM function in the reference section of this manual.

When used outside of pictures, TRANSFORM returns the identity matrix, which has no effect on a graphics display. Like other transformation functions, TRANSFORM affects only objects drawn with PLOT statements not GRAPH statements.

.4 Summary

The statements and topics described in this chapter include:

- Pictures (PICTURE subprograms)
- Invoking pictures with DRAW
- Invoking pictures with the transformation functions SHIFT, SCALE, ROTATE, SHEAR, and TRANSFORM
- Defining your own transformation matrices

So far, data for the examples has been supplied with DATA and READ statements or supplied as constants. The following chapter describes interactive means to supply graphics data.

Chapter 7

Graphics Input

VAX BASIC provides input statements so that a user[1] can interact with an application program. These statements enable a user to supply input to an application program at run time.

Many of the statements in this chapter can include an optional device identification number as a parameter. The device identification number is a number you supply to distinguish between devices when your program uses several devices. If you use a VAXstation, device identification numbers can be used to address different workstation windows. The programs in this chapter use only one device, or in the case of a VAXstation II, one workstation window; therefore, the statements do not include the optional device identification. For more information about using multiple devices, see Chapter 8.

The exact number and form of prompts, triggers, visual indicators, and echoes supported by each type of input varies with each device and is beyond the scope of this manual. Refer to the VAX GKS documentation and the hardware documentation for information about a particular device.

[1] In this chapter the term "user" is synonymous with a program operator; the word "you" refers to the programmer.

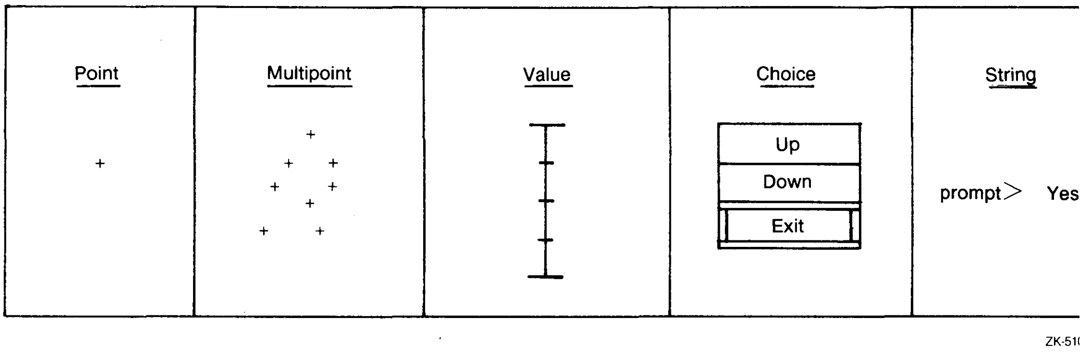
7.1 Input Types

There are five different types of graphics input described in the followin sections. For example, input could be a choice from a menu, the coordinates of a point, a series of points, a string, or a numeric value; you mus specify the type of input you want in your program. The five types of input are known in VAX BASIC as:

- CHOICE
- POINT
- MULTIPOINT
- STRING
- VALUE

Figure 7–1 illustrates the form of the different input types.

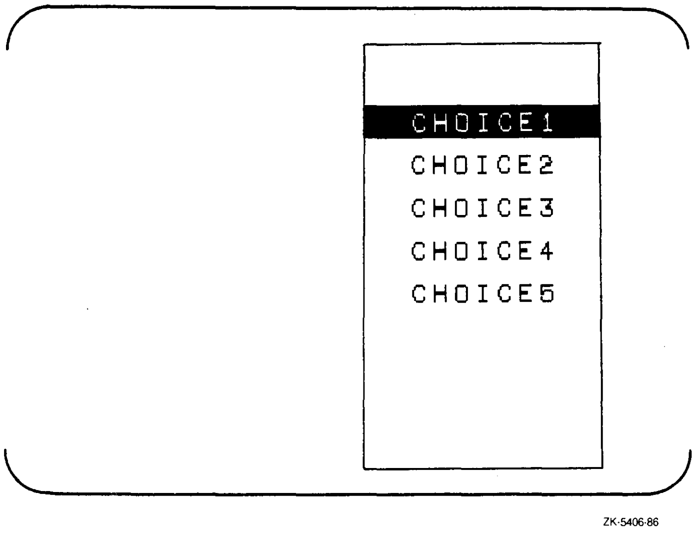
Figure 7–1: Types of Graphics Input



7.1.1 CHOICE Input

You request CHOICE data when you want a user to choose an item from a menu you supply. When CHOICE input is requested, the user is presented with a menu of items. The default response is emphasized in some way, usually with highlighting. The user can choose any item on the menu by moving the cursor in some manner. To select an item, the user presses the RETURN key, or possibly a mouse button. Figure 7–2 illustrates the default screen on a VT240 that is presented to a user when CHOICE input is requested.

Figure 7–2: The Default CHOICE Prompt on a VT240



.1.2 POINT Input

You request POINT input when you want a user to supply the position of a point. The user can supply a point in various ways, such as by positioning a cursor on a display surface with the keyboard arrow keys. The methods of entering points vary with each device.

When POINT input is requested, an initial point is displayed on the user's screen. The initial point is marked by a cross or some other visual marker; the actual marker varies with each device. On some devices it is possible to modify the prompt to your own design.

A user can move the prompt freely within the display area until the prompt marks the selected point. To enter the selected point as input, the user presses RETURN or performs a similar activity (this feature is device dependent).

7.1.3 MULTIPOINT Input

You request MULTIPOINT data when you want a user to supply a sequence of points.

When MULTIPOINT input is requested, an initial point, or series of points, is marked on the user's screen. The user can select this initial poir or choose other points. The ways to enter points vary with each device. For instance, on a VAXstation, points are entered automatically as the use moves the mouse. On a VT125, the user enters each point by pressing the space bar, then indicates when the input is complete by pressing RETURN. When the user moves the prompt to select a second point, a line is drawn connecting the first selected point to the second. A line is then drawn from the second point to the third, and so on until the series of points is complete. All of the points the user selects are connected by one continuous line on the screen.

7.1.4 STRING Input

You request STRING input when you want a user to supply an individua character or an entire text string to your program.

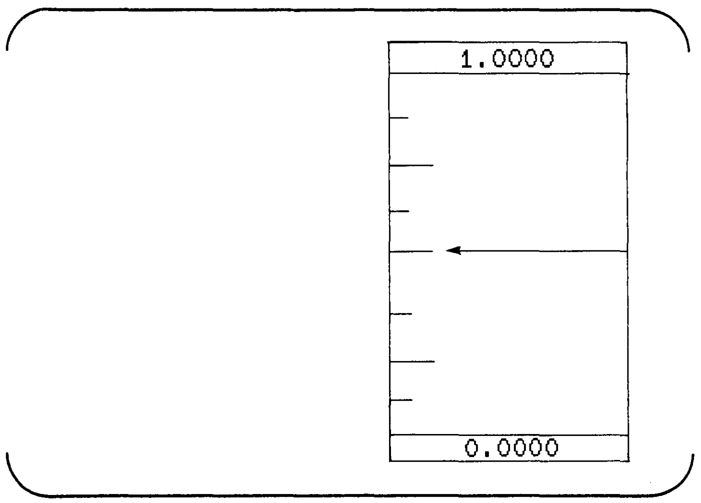
When STRING input is requested, the user is usually presented with an initial string. The user can accept the initial string, or enter a new string, or perhaps enter a null string if your program includes a default string response. Your program should include string handling functions tc analyze the user's input.

7.1.5 VALUE Input

You request VALUE data when you want a user to supply numeric data for your graphics program at run time.

When VALUE input is requested, the user is presented with a prompt similar to that illustrated in Figure 7–3. Depending on the device, the upper and lower limits of the acceptable range are displayed and an arrow is positioned to indicate the initial response. Some devices display the value that is currently selected. The user can select the initial value indicated or move the arrow along the scale to select another value withir the range presented. The user enters the selected value by pressing RETURN.

Figure 7–3: The Default VALUE Prompt on a VT240



The following sections discuss how your program accepts each of the graphics input types. To accept all types of input from a user, you can use the LOCATE statement. A single point can also be accepted with the GET statement, while a series of points can be accepted by the MAT LOCATE and MAT GET statements.

NOTE

> An optional UNIT clause can be included with many of the statements used in this chapter. This clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with the keyboard arrow keys or possibly with a mouse. Each of these methods of data entry is a different *unit*. The default unit number for each input class is 1. The default unit is used throughout this manual; therefore, the UNIT clause is not included in the examples in this chapter. For syntax rules related to the UNIT clause, see the particular input statement in the reference section of this manual. Some devices support more than one unit; for information about the units available on a particular device,

see the documentation for that device as well as the VAX GKS documentation.

7.2 CHOICE Input

Input statements for CHOICE allow you to:

- Set up an initial menu
- Set up an initial choice for the user
- Accept the user's actual choice

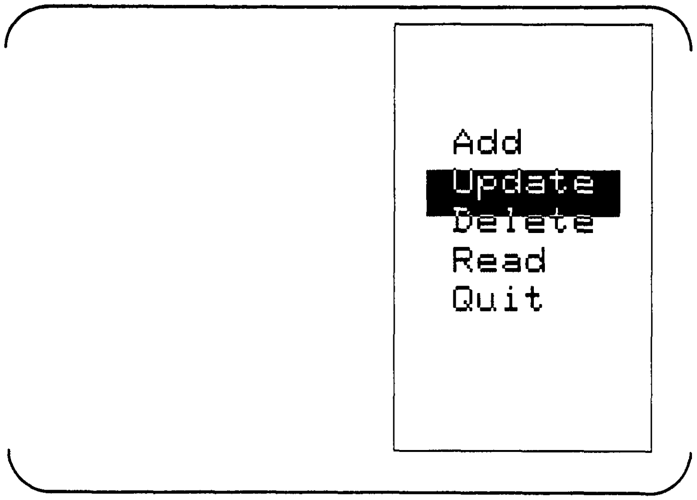
7.2.1 Setting the Initial Choice

The SET INITIAL CHOICE statement allows you to list the items you want displayed on the menu and set an initial menu selection for the use: The following statement sets up a menu with five possible selections and sets the initial selection to the second item in the menu. The menu is noi displayed until the LOCATE CHOICE statement is executed.

Example

```
DECLARE LONG task
SET INITIAL CHOICE                  &
            ,LIST ("Add"            &
                  ,"Update"         &
                  ,"Delete"         &
                  ,"Read"           &
                  ,"Quit")          &
                  : 2
Select_menu:
LOCATE CHOICE task
ON task GOTO Add_rtn,Update_rtn,Delete_rtn,Read_rtn,Quit &
             OTHERWISE Select_menu
    .
    .
    .
```

Output



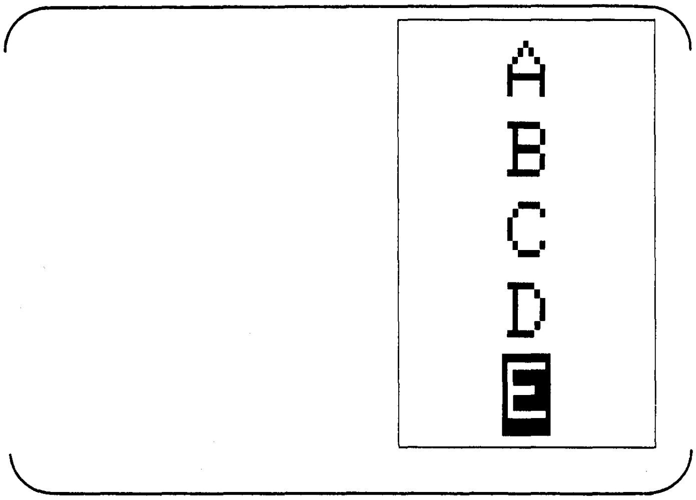
ZK-4944-86

The SET INITIAL CHOICE statement has an optional COUNT clause that can be used when you specify an array of strings rather than string expressions in the LIST clause. When you declare a string array, you need not display all the array elements as items in the menu; you can select a subset of the array elements starting with the first element in the array. Only the subset of items indicated by the COUNT clause will be displayed and the user cannot select an item in the menu that is not displayed on the screen. The following example illustrates the use of the COUNT clause and supplies an array of strings to the LIST clause.

Example

```
DECLARE LONG counter
DECLARE STRING which_letter
DIM STRING alphabet(65 TO 90)
alphabet(counter) = CHR$(counter) FOR counter = 65 TO 90
   .
   .
   .
SET INITIAL CHOICE, LIST alphabet                    &
                  , COUNT 5                          &
                  : 5
LOCATE CHOICE which_letter
   .
   .
   .
```

Output



ZK-4935-86

Subsequent requests for CHOICE input use the values specified in the SET INITIAL CHOICE statement unless you specify otherwise with another SET INITIAL CHOICE statement.

If you do not specify options in a LIST clause, VAX BASIC supplies the device dependent defaults. For VAXstation II workstations the default menu has two items: "Yes" and "No". VT125 and VT240 terminals provide a five-item menu by default with the options "CHOICE 1", "CHOICE 2", "CHOICE 3", "CHOICE 4", and "CHOICE 5".

'.2.2 Accepting CHOICE Input

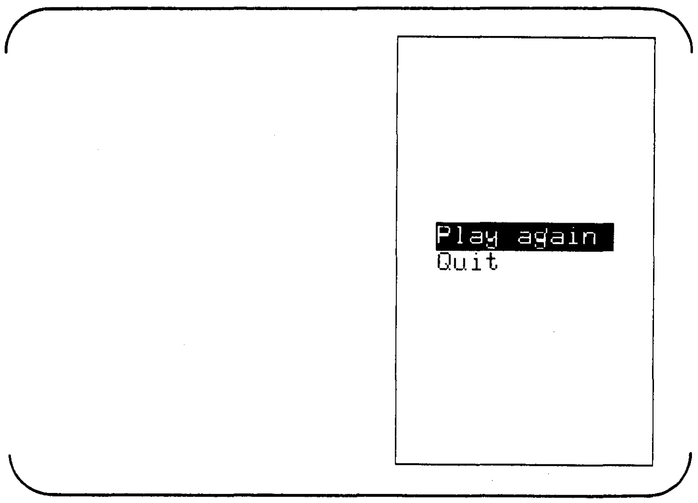
You accept CHOICE input with the LOCATE CHOICE statement. When the user makes a selection from a menu, an integer representing that item is assigned to a variable you supply with the LOCATE CHOICE statement. You set up the menu that is presented to the user with the SET INITIAL CHOICE statement; however, the menu is not displayed until the LOCATE CHOICE statement is executed.

In the following example the SET INITIAL CHOICE statement sets up a menu of two choices for the user: "Play again" or "Quit". When the LOCATE CHOICE statement is executed, the menu is displayed. "Play again" is highlighted as the initial choice; the user's actual choice is assigned to the integer variable *game_status* in the LOCATE CHOICE statement.

Example

```
DECLARE LONG game_status
SET INITIAL CHOICE , LIST ("Play again",      &
                          "Quit")             &
                         : 1
LOCATE CHOICE game_status
```

Output



ZK-4941-86

The initial choice presented to the user can be established in the SET INITIAL CHOICE statement, as shown in the previous example. You can override this initial choice with an optional INITIAL clause on the LOCATE CHOICE statement, as shown in the following example. The following example shows a longer menu and shows one method of dealing with the user choice in the program.

Example

```
PROGRAM psyche
  DECLARE LONG behavior
  Locate_choice:
  SET INITIAL CHOICE                                 &
              ,LIST ("withdrawn"                     &
                    ,"secretive"                     &
                    ,"reserved"                      &
                    ,"well-adjusted"                 &
                    ,"exuberant"                     &
                    ,"aggressive"                    &
                    ,"combative"                     &
                    ,"Quit program")                 &
                    : 3
  LOCATE CHOICE , INITIAL 4 : behavior

  SELECT behavior
    CASE = 1
           GRAPH TEXT AT 0,0.8 : "Do not leave this person alone - ever."
           GRAPH TEXT AT 0,0.7 : "Close curtains on rainy days. Recommend"
           GRAPH TEXT AT 0,0.6 : "constant care. Do NOT prescribe tranquilizers."

    CASE = 2
           GRAPH TEXT AT 0,0.8 : "Recommend intensive group therapy."
           GRAPH TEXT AT 0,0.7 : "Consider anti-depressant drugs."
           GRAPH TEXT AT 0,0.6 : "Confirm medical insurance."
    CASE = 3
           GRAPH TEXT AT 0,0.8 : "Recommend vigorous daily exercise."
           GRAPH TEXT AT 0,0.7 : "Continue group therapy."
    CASE = 4
           GRAPH TEXT AT 0,0.8 : "Make sure this person pays his/her bill,"
           GRAPH TEXT AT 0,0.7 : "then recommend ending therapy."
    CASE = 5
           GRAPH TEXT AT 0,0.8 : "Recommend relaxation techniques"
           GRAPH TEXT AT 0,0.7 : "and continue psychotherapy."
    CASE = 6
           GRAPH TEXT AT 0,0.8 : "Recommend intensive psychotherapy."
           GRAPH TEXT AT 0,0.7 : "Consider tranquilizers."
    CASE = 7
           GRAPH TEXT AT 0,0.8 : "Do not arm-wrestle with this person."
           GRAPH TEXT AT 0,0.7 : "Recommend sedation and hospitalization."
           GRAPH TEXT AT 0,0.6 : "Discontinue individual therapy unless"
           GRAPH TEXT AT 0,0.5 : "a guard is available at all times."
    CASE = 8
           EXIT PROGRAM
  END SELECT
END PROGRAM
```

7.3 POINT and MULTIPOINT Input

You accept POINT input with the LOCATE POINT and GET POINT statements, and MULTIPOINT input with the MAT LOCATE POINTS and MAT GET POINTS statements. The input statements for POINT and MULTIPOINT allow you to:

- Set up an initial point or series of points for the user
- Accept the user's actual input

7.3.1 Accepting Points with the Default Transformation

This section describes how to accept input points from a user when transformation 1, the default transformation, is the only transformation in your program. Transformation 1 is the only transformation when no additional transformations are defined in your program, and when no SET VIEWPORT or SET WINDOW statements are executed other than for redefining the default transformation.

The LOCATE POINT statement accepts a point the user enters at the terminal. After execution of the following statement, the world coordinates of the user-supplied point are contained in *x_coord* and *y_coord.*

```
LOCATE POINT x_coord, y_coord
```

The MAT LOCATE POINTS statement accepts a series of points that the user enters at the terminal into two arrays you have supplied with the statement: one array of x-coordinates, and one of y-coordinates. After execution of the following statement, the world coordinates of the user-supplied points are contained in the arrays *x_coords* and *y_coords.*

```
DIM SINGLE x_coords(20), y_coords(20)
    .
    .
    .
MAT LOCATE POINTS x_coords, y_coords
```

After the prompt is displayed, the user can accept the initial point or enter other points. The methods of entering each point vary with each device. For example, with a VT240, a user enters each point by pressing the space bar, then indicates when the series of points is complete by pressing RETURN.

You can specify the world coordinates of an initial point with the optional AT clause in both the LOCATE POINT and MAT LOCATE POINTS statements. The following statement displays a prompt at the initial position 0.2,0.5. The user can accept this initial point, or move the prompt and enter the position of another point. After execution, the world coordinates of the user-supplied point are assigned to *x_coord* and *y_coord*.

```
LOCATE POINT , AT 0.2,0.5 : x_coord, y_coord
```

See Section 7.3.5 for details on how to present a series of points to the user.

An optional COUNT clause in the MAT LOCATE POINTS statement allows you to retrieve the number of points actually supplied by the user. In the following example, the arrays are set up to hold 26 coordinate values; the actual number of points supplied by the user is assigned to the variable *how_many*.

If the user inputs fewer points than the number of elements in the coordinate arrays and you plan to use these coordinates for output, you should use the COUNT clause to specify the number of array elements to be used for the display.

If a user supplies more points than the number of elements in the arrays, the extra points are discarded by VAX BASIC. For instance, in the following example, if a user enters 30 points, the last four points will be discarded. The following example checks whether the user has entered extra points. If this is the case, the variable supplied with the COUNT clause in the output statement is adjusted to avoid referencing array elements that do not exist.

Example

```
DECLARE LONG CONSTANT max_points = 26
DECLARE LONG how_many
DIM SINGLE xs(1 TO max_points),ys(1 TO max_points)
GRAPH TEXT AT 0,0.9 : "Enter up to " + STR$(max_points) + " points"
MAT LOCATE POINTS , AT .10,.15                &
                  , COUNT how_many            &
                  : xs, ys
IF how_many > max_points
   THEN how_many = max_points
        GRAPH TEXT AT 0,0.75: "You have entered too many points."
        GRAPH TEXT AT 0,0.65: "The extra points have been discarded."
END IF
SET POINT COLOR 2%
MAT GRAPH POINTS , COUNT how_many             &
                 : xs, ys
END
```

7.3.2 Accepting Points with Multiple Transformations Defined

When possible, VAX BASIC maps the coordinates of an input point to world coordinates using the transformation that is defined for the viewport the point falls into. The following sections describe how VAX BASIC determines which transformation to use under the following circumstances:

- When input points map to an area of NDC space where a single viewport is valid
- When input points map to an area of NDC space where viewports overlap

7.3.2.1 Accepting Points That Map to One Valid Viewport

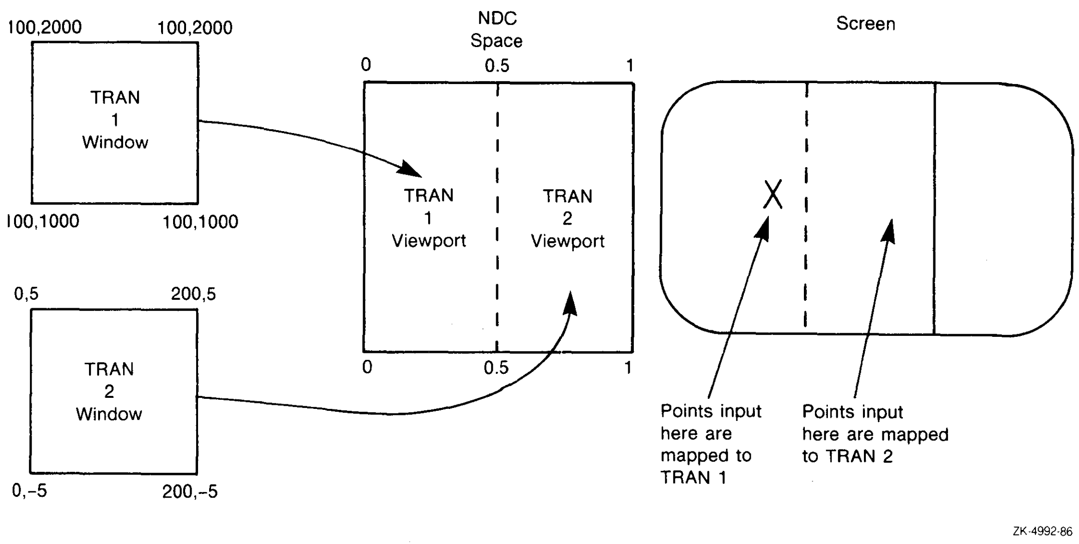
When a user enters a point as input, VAX BASIC determines the device coordinates of the point entered, then maps the point to NDC space. If the input point maps to an area of NDC space where a single viewport is valid, the transformation associated with that viewport is used to determine the world coordinates of the point.

In the following example, two transformations are established. Transformation 1 redefines the default world viewport and sets the window to -100,100,1000,2000. Transformation 2 establishes an additional viewport and sets the window to 0,200,-5,5. Notice that these viewports do not overlap and the two viewports together include all of NDC space. When a point is input, the point falls into the viewport for either transformation 1 or 2 and is interpreted according to the transformation associated with the appropriate viewport. Figure 7–4 illustrates these two transformations and shows an input point that is interpreted according to the viewport for transformation 1. The resulting world coordinates held in *x* and *y* would be 50,1500.

Example

```
DECLARE SINGLE x,y
SET WINDOW , TRAN 1 : -100,100,1000,2000
SET VIEWPORT , TRAN 1 : 0,0.5,0,1
SET WINDOW , TRAN 2 : 0,200,-5,5
SET VIEWPORT , TRAN 2 : 0.5,1,0,1
    .
    .
    .
LOCATE POINT x,y
    .
    .
    .
```

igure 7–4: An Input Point is Mapped to the Underlying Viewport



You can supply an optional integer variable with the LOCATE POINT statement to retrieve the transformation number VAX BASIC has used to interpret the user-supplied coordinates. For example, the following statement instructs VAX BASIC to use the point 0.5,0.5 as the initial point. When the user enters a point, the coordinates are assigned to *x_coord* and *y_coord*. The transformation used to interpret these device coordinates into world coordinates is assigned to *which_tran*. This is useful if you have multiple transformations defined and you want to confirm which transformation was actually used.

Example

```
DECLARE SINGLE x_coord,y_coord,                               &
        LONG which_tran
SET VIEWPORT , TRAN 1 : 0,0.5,0,1
SET VIEWPORT , TRAN 2 : 0.5,1,0,1
    .
    .
    .
LOCATE POINT , AT 0.5,0.5                                     &
             : x_coord, y_coord ,  which_tran
    .
    .
    .
```

Note that when transformation 1 has been redefined with a viewport other than 0,1,0,1 (all of NDC space), it is possible for points to be outsid any defined viewport. When this happens, VAX BASIC signals an error. In the following example, transformation 1 is redefined to be a portion of NDC space. This leaves the remaining portion of NDC space with no transformation defined, and it is possible for a user to supply a point that falls beyond the boundaries of any defined transformation.

Example

```
SET VIEWPORT , TRAN 1 : 0.5,1,0.5,1
    .
    .
    .
MAT LOCATE POINTS x_coord, y_coord
```

The optional USING TRAN clause on the LOCATE POINT statement lets you specify the transformation VAX BASIC should use to display the initial point. If you do not include a USING TRAN clause, the current transformation is used. In the following example, transformation 2 is the current transformation when the LOCATE POINT statement is executed. However, the USING TRAN clause specifies that transformatio 1 should be used to display the initial point regardless of the fact that transformation 2 is the current transformation. The USING TRAN clause affects only the display of the initial point; the user-supplied coordinates are transformed with the current transformation.

Example

```
DECLARE SINGLE x,y
SET VIEWPORT , TRAN 1 : 0.5,1,0.5,1
SET VIEWPORT , TRAN 2 : 0,0.5,0,0.5
LOCATE POINT , AT 0.6,0.8 USING TRAN 1 : x,y
```

'.3.2.2 Accepting Points That Map to Overlapping Viewports

When you do not change the default viewport but do define other transformations with the SET WINDOW statement, these transformations all map onto the entire NDC space. For example, the following transformations all share the same viewport.

Example

```
SET WINDOW , TRAN 1 : 0,100,0,100
SET WINDOW , TRAN 2 : -1, 1, -1, 1
SET WINDOW , TRAN 3 : 50, 200, 0, 5
    .
    .
    .
```

In this case, VAX BASIC cannot distinguish which transformation to use on the basis of the viewport alone; therefore, it refers to a list of established transformations and selects the transformation with the *highest input priority.*

When more than one transformation is defined, VAX BASIC maintains a prioritized list of transformations to use to translate POINT and MULTIPOINT input when viewports overlap. At the start of program execution, transformation 1 (defining the entire NDC space) is the only transformation on the list; therefore, at the start of program execution, transformation 1 is also the transformation with the highest input priority. Points input at the start of program execution are interpreted with transformation 1.

When a subsequent transformation is set with a SET TRANSFORMATION, SET WINDOW, or SET VIEWPORT statement, this new transformation automatically becomes the transformation at the top of the list and, therefore, acquires a higher input priority than transformation 1. The transformation with the highest input priority is also the current transformation for output unless a SET INPUT PRIORITY statement has been executed (see Section 7.3.3).

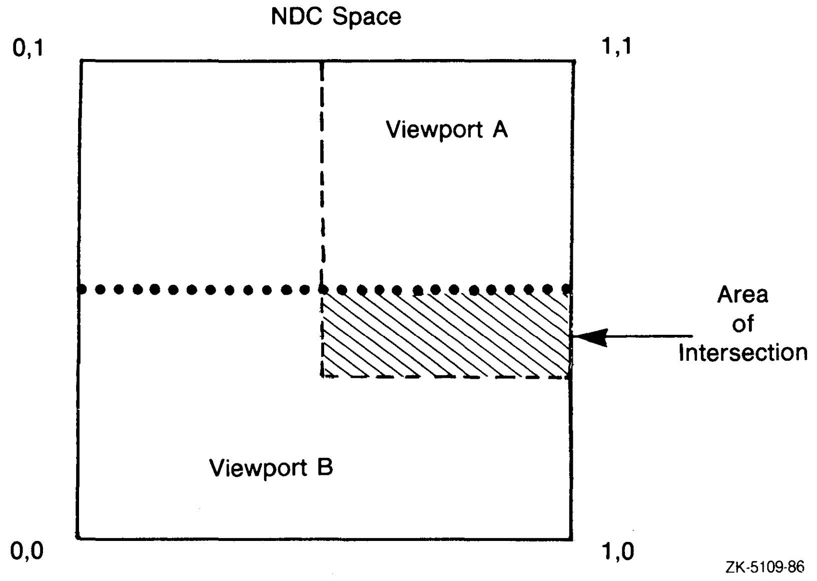
In the following example, the point 50,50 is the initial point and is displayed on the screen using transformation 1 because it is the transformation specified in the USING TRAN clause. When the user enters each point, the coordinates are assigned to the next elements in the arrays *x_coords* and *y_coords.* Any point entered by the user falls into the default viewport. As no other viewports are defined, the default viewport is the viewport for both transformation 1 and transformation 2. The transformation used to interpret these coordinates into world coordinates is the transformation with the highest input priority (in this case, transformation 2, because this is the transformation most recently set). The actual transformation used to interpret the user-supplied points in the following example is assigned to *which_tran,* the integer variable supplied with the MAT LOCATE POINTS statement.

Example

```
DECLARE LONG which_tran
DIM SINGLE x_coords(12),y_coords(12)
SET WINDOW , TRAN 1 : 0,100,0,100
SET WINDOW , TRAN 2 : 0,25,0,25
    .
    .
    .
MAT LOCATE POINTS , AT 50,50 USING TRAN 1                 &
             : x_coords, y_coords,  which_tran
    .
    .
    .
```

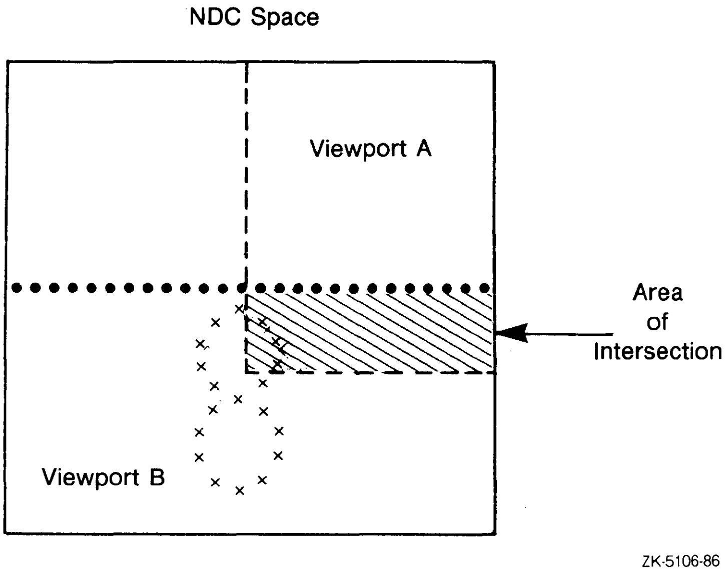
Overlapping viewports occur only when a transformation other than transformation 1 is defined. When possible, VAX BASIC maps the coordinates of an input point to world coordinates using the transformation that is defined for the viewport the point falls into. When viewports overlap, a point can fall into more than one viewport and VAX BASIC uses the transformation with the highest input priority that also contains the point. In Figure 7–5, a user-supplied point can fall into the area of intersection of both viewports. As both viewports are valid, VAX BASIC selects the viewport with the highest input priority to interpret the input.

Figure 7–5: Interpreting POINT Input When Viewports Overlap



When the user supplies a series of points (MULTIPOINT input), VAX BASIC identifies the viewports that contain *all* of the points. If there is no conflict and all of the points are contained within one viewport, then that transformation is used. However, when all of the points are contained in more than one viewport, the transformation with the highest input priority is used. In Figure 7–6, the series of points shown is interpreted with the transformation defined for Viewport B.

Figure 7–6: Interpreting MULTIPOINT Input When Viewport Overlap



Note that when transformation 1 has been redefined with a viewport othe than 0,1,0,1, it is possible for points to be outside any defined viewport. When this happens, VAX BASIC signals an error. Similarly, VAX BASIC signals an error when no single viewport contains all of a series of points

The following example defines several overlapping transformations. The list of input priorities changes whenever a new transformation is defined, when the SET TRANSFORMATION statement is executed, or when the SET INPUT PRIORITY statement is executed. At various points in the program POINT input is requested from a user. The actual transformation used cannot be determined until the position of the input point is indicated. Remember that VAX BASIC uses the priority list only when viewports overlap; if no overlapping viewports are defined, the viewport that contains the points is automatically used to transform the input coordinates into world coordinates. The number of the actual transformation VAX BASIC used is assigned to the variable *which_tran*.

Example

```
DECLARE LONG which_tran,                                  &
        SINGLE x1,y1,x2,y2,x3,y3,x4,y4
!+
!TRAN 1 has highest priority at start of program execution
!-
SET WINDOW , TRAN 2 : 0,10,0,10 !TRAN 2 now has highest input priority
SET WINDOW , TRAN 3 : -1,1,-1,1 !TRAN 3 now has highest input priority
SET TRANSFORMATION 2            !TRAN 2 now has highest input priority
!+
!Tran 2 has the highest priority
!VAX BASIC assigns the transformation it used to interpret the point
!x1,y1 to which_tran
!-
LOCATE POINT x1,y1 , which_tran
SET VIEWPORT , TRAN 3 : 0.5,1,0.8,1  !TRAN 3 now has highest input priority
!+
!Tran 3 has the highest priority
!VAX BASIC assigns the transformation it used to interpret the point
!x2,y2 to which_tran
!-
LOCATE POINT x2,y2 , which_tran
SET WINDOW , TRAN 4 : 0,0.5,0,0.5    !TRAN 4 now has highest input priority
!+
!Tran 4 has the highest priority
!VAX BASIC assigns the transformation it used to interpret the point
!x3,y3 to which_tran
!-
LOCATE POINT x3,y3 , which_tran
SET TRANSFORMATION 1                 !TRAN 1 now has highest input priority
!+
!Tran 1 has the highest priority
!VAX BASIC assigns the transformation it used to interpret the point
!x4,y4 to which_tran
!-
LOCATE POINT x4,y4 , which_tran
   .
   .
   .
```

'.3.3 Changing the Input Priority

The SET INPUT PRIORITY statement allows you to change the order of the transformations on the input priority list. You use the greater than (>) or less than (<) sign to reorder the priorities. For instance, in the following example, the SET INPUT PRIORITY statement makes 3 the transformation with the highest priority for input regardless of the SET TRANSFORMATION 4 statement. Note that the SET INPUT PRIORITY statement has no effect on the current transformation for displaying output; in this case, transformation 4 remains the current transformation for displaying output.

Example

```
DECLARE SINGLE x,y,             &
        LONG which_tran
   .
   .
   .
SET TRANSFORMATION 4       !TRAN 4 now has the highest input priority
SET INPUT PRIORITY 3 > 4  !TRAN 3 now has a higher input priority than 4
!+
!Point x,y is interpreted with the transformation
!assigned to which_tran
!-
LOCATE POINT x,y , which_tran
   .
   .
   .
```

If the point x,y is contained in the viewports for both transformation 3 and transformation 4, VAX BASIC uses the transformation with the highest input priority—now transformation 3. Without this SET INPUT PRIORITY statement, VAX BASIC would transform the point according to the definition of transformation 4, assuming that the point was contained in that viewport.

7.3.4 Setting the Initial Point

You can select an initial point with the LOCATE POINT and MAT LOCATE POINT statements, as shown in previous sections. Alternatively, you can set the initial point with the SET INITIAL POINT statement. If you have more than one transformation defined, you can specify which one VAX BASIC should use to display this point by including an optional USING TRAN clause. If you do not include a USING TRAN clause, VAX BASIC uses the current transformation to display output.

The following example instructs VAX BASIC to display the world coordinate point 850,150 using the transformation previously defined as 2 in the program. This point is displayed as the initial prompt at the appropriate position on the display surface when input is requested. Note that the current transformation is 3 when the SET INITIAL POINT statement is executed.

Example

```
    .
    .
    .
SET WINDOW , TRAN 2 : 0,1000,0,500
SET VIEWPORT , TRAN 2 : 0.0,0.75,0.0,0.5
SET WINDOW , TRAN 3 : -100,100,-100,100
SET VIEWPORT , TRAN 3 : 0.5,1,0.0,0.5
SET INITIAL POINT, USING TRAN 2 : 850,150
    .
    .
    .
LOCATE POINT x_user,y_user
    .
    .
    .
```

.3.5 Setting an Initial Series of Points

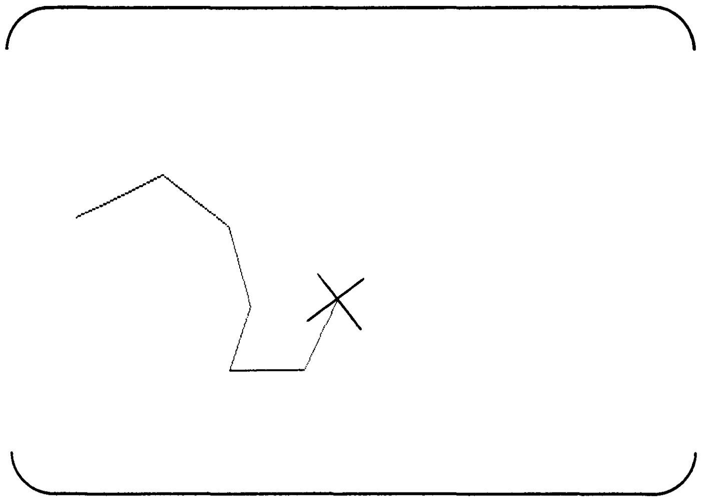
The MAT LOCATE POINTS and MAT GET POINTS statements allow you to present the user with one initial point specified in the optional AT clause; the SET INITIAL MULTIPOINT statement allows you to present a user with a series of points. You supply initial MULTIPOINT data as two arrays of world coordinates: an array of x-coordinates and an array of y-coordinates. If you have more than one transformation defined, you can specify in the optional USING TRAN clause which transformation VAX BASIC should use to interpret these initial points. If you do not supply an alternative transformation, VAX BASIC uses the current output transformation to display the initial points on the screen.

An optional COUNT clause can be specified with this statement to indicate the number of elements from each array to be used for the display. In the following example, two arrays of coordinates are declared with 10 elements each. However, the SET INITIAL MULTIPOINT statement uses a COUNT clause to limit the initial MULTIPOINT data to the first seven elements of each array (elements 0 through 6). Without the COUNT clause, all points in the arrays would be displayed. The output shows how this initial set of points is displayed on a VT240. (On some devices, the DELETE key can be used to delete the initial series of points.)

Example

```
DECLARE LONG counter, how_many
DIM SINGLE x_stars(9),y_stars(9),          &
           user_x(40), user_y(40)
DATA 10,50, 30,60, 45,47.5, 50,29, 45,14,  &
     62.5,14, 70,30, 80,19, 82,22, 89,25
READ x_stars(counter),y_stars(counter) FOR counter = 0% TO 9%
   .
   .
   .
SET WINDOW , TRAN 1 : 0,100,0,100
!+
!Display only seven of the points
!-
SET INITIAL MULTIPOINT, COUNT 7            &
                      : x_stars,y_stars
MAT LOCATE POINTS , COUNT how_many : user_x, user_y
   .
   .
   .
```

Output



Note that no USING TRAN clause is included in this statement; therefor
VAX BASIC transforms the seven points using the current transformatior
The user can select this initial diagram or move the prompt to select oth
points. In this case, the user can supply up to 41 points.

3.6 Accepting Points Within Pictures

When points are accepted during the execution of a picture, the points accepted by the LOCATE POINT and MAT LOCATE POINTS statements are not affected by the picture transformations specified on the DRAW statement that invoked the picture. If you want the input points to be affected by these transformations, you should use the GET POINT and MAT GET POINTS statements. The coordinates accepted by a GET POINT or a MAT GET POINTS statement are transformed according to the *inverse* of the transformations specified on a DRAW statement; that is, the device coordinates of input points are transformed to world coordinates and then transformed according to the applied transformation functions. For instance, if a picture is invoked with SCALE(2) and points are input during picture execution, the input points will be transformed to world coordinates according to the current transformation, and then divided by two (the inverse of SCALE(2)).

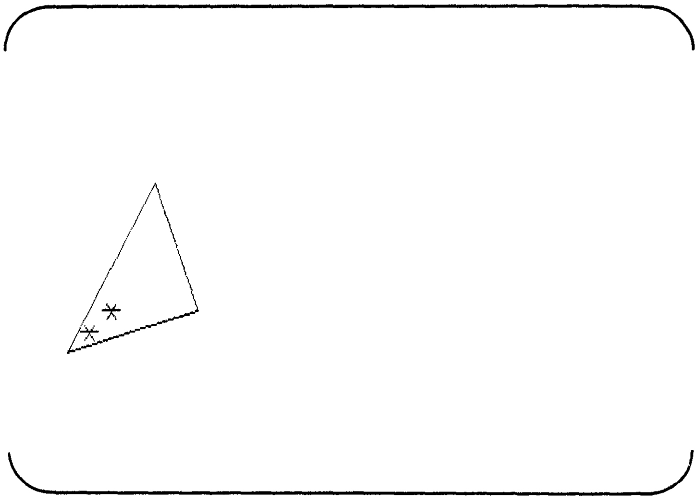
The following example uses the GET POINT and LOCATE POINT statements in a picture definition. The first screen shows the picture drawn without any transformation functions. For the second screen, the picture is drawn with the SHIFT function. Notice that the coordinates of the point accepted with the LOCATE POINT statement are not shifted.

Example

```
PROGRAM get_the_point
  OPTION TYPE = EXPLICIT
  EXTERNAL PICTURE loc_get_point
Screen_1:
  DRAW loc_get_point
  SLEEP 5%
  CLEAR
Screen_2:
  DRAW loc_get_point WITH SHIFT(0.3,0)
END PROGRAM
```

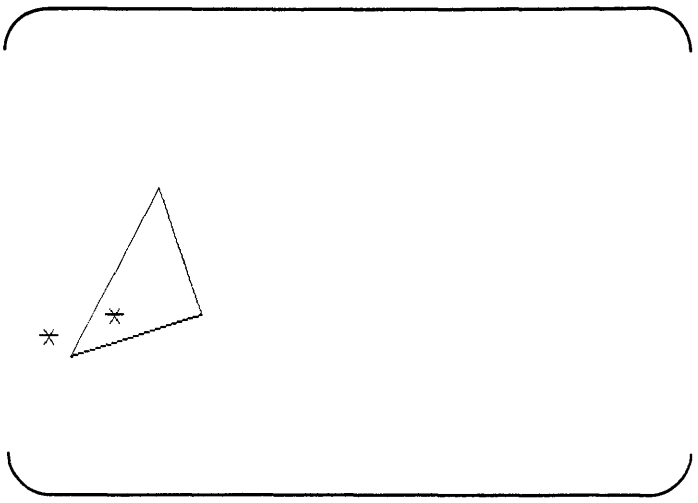
```
PICTURE loc_get_point
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE x_coord,y_coord,lx_coord,ly_coord
  PLOT LINES 0.1,0.2; 0.4,0.3;               &
               0.3,0.6; 0.1,0.2
  GET POINT x_coord, y_coord
  PLOT POINTS x_coord, y_coord
  LOCATE POINT lx_coord, ly_coord
  PLOT POINTS lx_coord, ly_coord
END PICTURE
```

Output Screen 1:



ZK-4939-86

Output Screen 2:



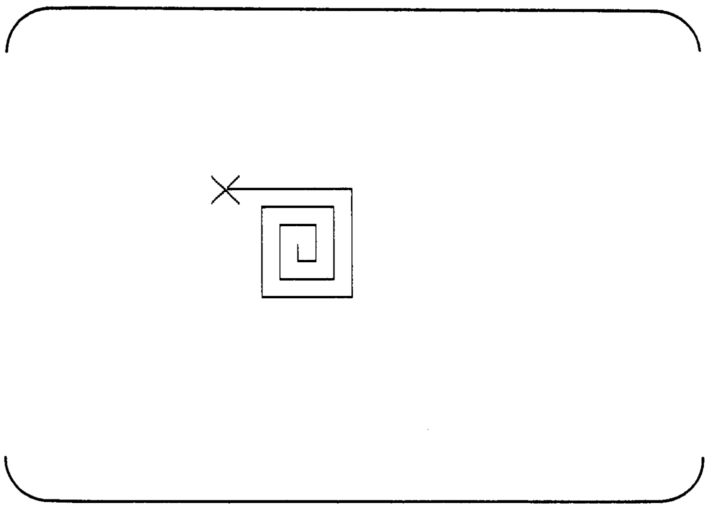
ZK-4938-86

Similarly, when MULTIPOINT input is accepted within a picture definition, the points accepted by the MAT LOCATE POINTS statement are not affected by the transformation functions specified on the DRAW statement that invoked the picture, while points accepted by the MAT GET POINTS statement are affected. The device coordinates of input points are transformed to world coordinates and then transformed according to the applied transformation functions. The following examples illustrate the difference between the MAT GET POINTS statement and the MAT LOCATE POINTS statement used within a picture. The first picture accepts user input with the MAT LOCATE POINTS statement while the second picture uses the MAT GET POINTS statement. For purposes of illustration, the examples assume the user input is the same for both pictures.

Example

```
PICTURE locate_doodle
  DECLARE LONG CONSTANT max_pts = 26
  DECLARE SINGLE  x_array(1 TO max_pts),y_array(1 TO max_pts)   &
          , LONG how_many
  MAT LOCATE POINTS                                             &
              , AT 0,0                                          &
              , COUNT how_many                                  &
              : x_array, y_array
  IF how_many > max_pts
     THEN how_many = max_pts
  END IF
  MAT GRAPH LINES , COUNT how_many : x_array,y_array
END PICTURE

PICTURE get_doodle
  DECLARE LONG CONSTANT max_pts = 26
  DECLARE SINGLE x_array(1 TO max_pts),y_array(1 TO max_pts)   &
          , LONG how_many
  MAT GET POINTS                                               &
          , AT 0,0                                             &
          , COUNT how_many                                     &
          : x_array, y_array
  IF how_many > max_pts
     THEN how_many = max_pts
  END IF
  MAT GRAPH LINES , COUNT how_many : x_array,y_array
END PICTURE
```

User Input:



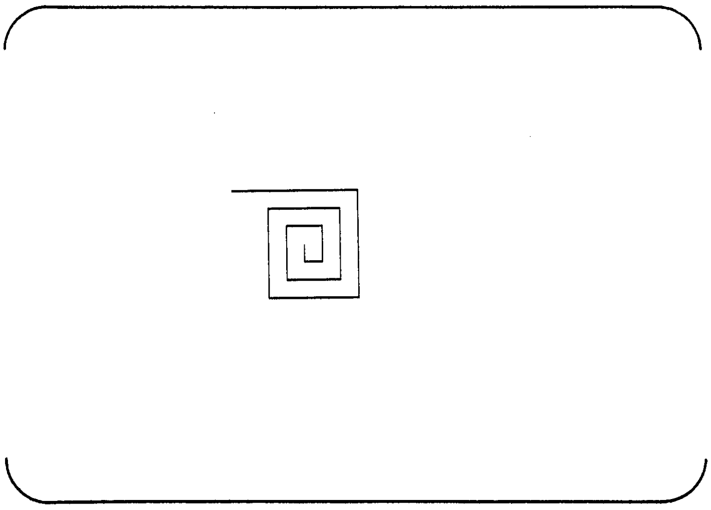
ZK-5528-86

When the pictures are invoked, only the points retrieved by the MAT GET POINTS statement are affected by the transformation functions in the DRAW statement. Compare the output of the DRAW statements in the following example.

Example

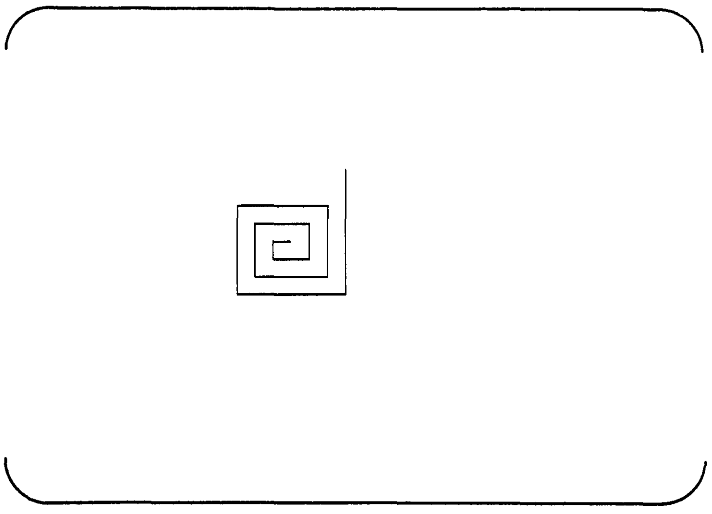
```
PROGRAM compare
   OPTION TYPE = EXPLICIT
   OPTION ANGLE = DEGREES
   EXTERNAL PICTURE locate_doodle,get_doodle
   !+
   !Set up point of origin at center for rotation
   !-
   SET WINDOW -50,50,-50,50
   DRAW locate_doodle WITH ROTATE(90)   !Set up screen 1
   SLEEP 5%
   CLEAR
   DRAW get_doodle WITH ROTATE(90)      !Set up screen 2
END PROGRAM
```

Output Screen 1:



ZK-5529-86

Output Screen 2:



ZK-5527-86

7.4 STRING Input

Input statements for STRING allow you to:

- Set up an initial string for the user
- Accept the user-supplied string

7.4.1 Accepting STRING Input

You accept STRING input with the LOCATE STRING statement. Execution of the LOCATE STRING statement assigns the user-supplied string to a string variable you supply. For example:

```
DECLARE STRING user_supplied_string
    .
    .
    .
LOCATE STRING user_supplied_string
```

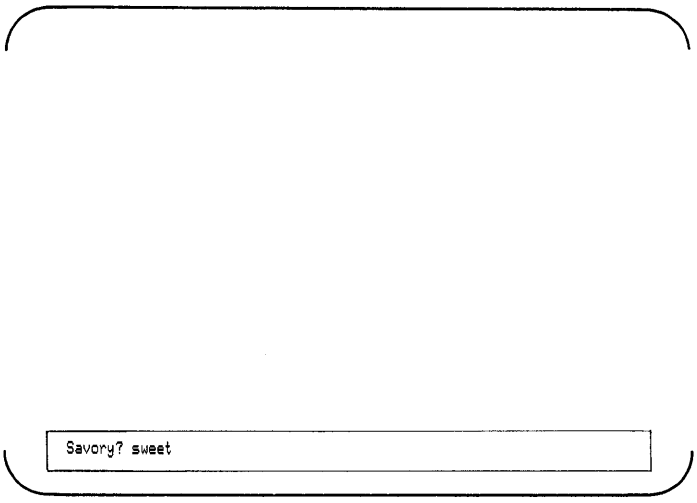
The LOCATE STRING statement has an optional INITIAL clause. You can use this clause to present an initial string to the user. Depending on the device, the cursor is placed at the end of the initial string, and the user can either press the RETURN key to accept the initial string or enter another string. The user-supplied string is concatenated to the initial string. However, depending on the device, the user may be able to delete the initial string by pressing the DELETE key or CTRL/U. You can detect the absence of the initial string in your program.

You can use various string handling functions to examine the user-supplied string. Depending on the the contents of the examined string, your program can branch accordingly. In this example, the user has responded with the string "sweet". After execution of the LOCATE STRING statement, *taste_choice* contains the string "Savory?sweet".

Example

```
DECLARE STRING taste_choice
LOCATE STRING, INITIAL "Savory?" : taste_choice
    .
    .
    .
```

Output



```
Savory? sweet
```

ZK-5235-86

You can use the null string to indicate a default response, as shown in the following example. The following example sets the initial string to "Save drawing [yes]?" informing the user that pressing the RETURN key will be interpreted as a positive response. In your program, you can branch to different routines according to the user-supplied string. This example presents the initial string as a prompt; unless the user has deleted all or part of the initial string, the resulting string contains both the prompt and the user-supplied response.

Example

```
EXTERNAL PICTURE save_it,menu
DECLARE STRING CONSTANT initial_prompt = "Save drawing [yes]? "
DECLARE STRING initial_prompt, whole_string
LOCATE STRING , INITIAL initial_prompt : whole_string
!+
!Check if user responded with a <CR>
!-
IF whole_string = initial_prompt
   THEN DRAW save_it
   ELSE DRAW menu
END IF
   .
   .
   .
```

.4.2 Setting the Initial String

You can set the initial string with the SET INITIAL STRING statement as an alternative to using the INITIAL clause with the LOCATE STRING statement. The following statement sets the initial string to "Sooner or later [sooner]?". This informs the user how a default response will be interpreted.

```
SET INITIAL STRING "Sooner or later [sooner]? "
```

Subsequent requests for STRING input use the initial string specified in the most recent SET INITIAL STRING statement unless this is overridden by an INITIAL clause in a LOCATE STRING statement.

Note that unless you set an initial string in your program with a SET INITIAL STRING statement or with an INITIAL clause on the LOCATE STRING statement, no string is presented to the user.

7.5 VALUE Input

Input statements for VALUE allow you to:

- Set up an initial range of acceptable values
- Set up an initial value for the user
- Accept the user's actual input

7.5.1 Accepting VALUE Input

You accept VALUE input with the LOCATE VALUE statement. With this statement you can optionally declare an acceptable range of values and indicate an initial selection. When you do not specify an initial value and range, a default range of 0 through 1 and an initial value of 0.5 are used. The following example changes the value prompt displayed to the user with a LOCATE VALUE statement.

Example

```
DECLARE SINGLE real_value
LOCATE VALUE , RANGE 5 TO 10    &
             , INITIAL 8.5      &
             : real_value
    .
    .
    .
```

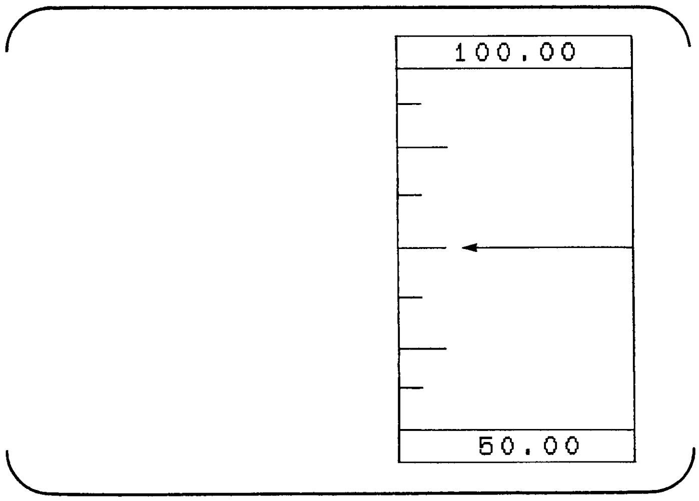
7.5.2 Setting the Initial Value

The SET INITIAL VALUE statement allows you to set the range of accept able values and select an initial value to be presented to the user. You can use this statement as an alternative to using the INITIAL clause on the LOCATE VALUE statement. The initial default range of 0 through 1 is used when you do not specify an alternative range. The acceptable range is displayed as the prompt to the user; the form of the display varie with each device. On a VT240, the display is presented in the form of a continuous line with the lowest acceptable value at the bottom and the highest acceptable value at the top. This range and initial value are also used for any subsequent requests for VALUE input. For example, the following SET INITIAL VALUE statement declares that acceptable values are within the range 50 through 100 and sets the initial value to 75.

Example

```
DECLARE SINGLE CONSTANT first = 50,        &
                        last = 100
DECLARE SINGLE how_much
    .
    .
    .
SET INITIAL VALUE, RANGE first TO last : 75
    .
    .
    .
LOCATE VALUE how_much
```

Output



ZK-5231-86

An initial value specified in a LOCATE VALUE statement overrides the initial value set in the SET INITIAL VALUE statement. However, if no INITIAL clause is included in a LOCATE VALUE statement, subsequent requests for VALUE input use the last initial value set with a SET INITIAL VALUE statement.

The initial value must lie within the current range. When this is not the case, VAX BASIC signals the run-time error ILLINIVAL, "illegal initial value" (ERR = 284).

7.6 Changing Echo Areas

The *echo area* is the portion of your screen where the prompt appears and where input can be accepted from a user. It can be all or part of the screen. So far in this chapter, the default echo area has been displayed. The default boundaries of the echo area for each type of input are device dependent.

You can change the echo boundaries in your application to a preferred area of your screen for the CHOICE, STRING, and VALUE echo areas.

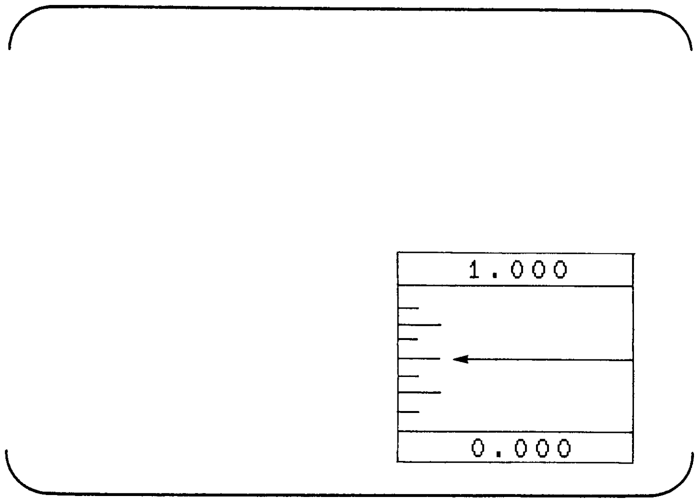
Use the SET . . . ECHO AREA statements to change the default echo are You must specify the boundaries of an echo area in device coordinates in the order left, right, bottom, top. Because the echo boundaries must be in device coordinates, VAX BASIC supplies statements that allow you to retrieve the current echo area boundaries. You can retrieve the device coordinates of a particular device with the ASK DEVICE SIZE statement. This statement is discussed in Chapter 8. To retrieve the boundaries of the current echo area on a particular device, use the appropriate ASK . . . ECHO AREA statement. Boundaries for VAXstations and VT125 and VT240 terminals are listed in Appendix B.

The following example retrieves the data for the default VALUE echo are on a VT125, then uses this data to reduce the echo area. You can alter the default echo area for STRING and CHOICE input in a similar fashion Further examples are given in the reference section of this manual.

Example

```
DECLARE SINGLE xmin,xmax,ymin,ymax, user_val
ASK VALUE ECHO AREA xmin,xmax,ymin,ymax
SET VALUE ECHO AREA xmin,xmax,ymin,ymax/2
LOCATE VALUE user_val
    .
    .
    .
```

Output



ZK-5229-86

.7 Summary

An interactive graphics program can:

- Set initial input data, such as the options in a menu
- Accept input from a user, such as the world coordinates of points
- Define the input priority when viewports overlap
- Change defaults, such as the echo area

The following chapter shows how to use the input statements discussed here when you use more than one device.

Chapter 8

Advanced Graphics Programming Techniques

A complex graphics application can open several different devices for simultaneous input and output, store graphics data in files, and display the stored data. This chapter shows you how to:

- Use alternate or multiple devices
- Send graphics output to a data file rather than an output device
- Display the contents of graphics data files
- Determine the capabilities of various supported devices
- Define device transformations

ı.1 Using Alternate or Multiple Devices

To use more than one device, you must include the OPEN . . . FOR GRAPHICS statement in your program. For example, the following statement opens a device and identifies it as device #2. The quoted string must be a system name or logical name for an actual device. (The quoted string can also be a file name, as discussed in Section 8.2.)

```
OPEN "VTA247:" FOR GRAPHICS AS DEVICE #2
```

Normally on VAX/VMS systems, all terminals other than your own are protected from you. In order to open a terminal other than your own, you must have SYSPRV privileges; without these privileges, you can access another terminal only if a privileged user issues the DCL command SET PROTECTION/DEVICE for that terminal. This command can be included in the installation startup file to make such a terminal permanently available to you. If you require changes to your privilege status, see your system manager.

The OPEN . . . FOR GRAPHICS statement is optional. If you do not explicitly open a device, VAX BASIC opens the default device for you. Th default device is opened with the equivalent of the following statement:

```
OPEN "" FOR GRAPHICS AS DEVICE #0
```

When you do not identify a device with an OPEN . . . FOR GRAPHICS statement, VAX BASIC automatically uses the default identification of 0, which is equivalent to both SYS$INPUT and SYS$OUTPUT. The default identification is adequate when you are using only one device, such as when you both create a program and display the image on a single video display unit; however, you need device identification numbers when you use more than one device in an application.

Note that multiple workstation windows on a VAXstation should be identified as separate devices; otherwise, all output is displayed in the same window.

8.1.1 The Device Identification Clause

The device identification number is an integer that you assign to distinguish between different devices in your program. The number is explicitly assigned to a device in the device identification clause in the OPEN . . . FOR GRAPHICS statement.[1]

Many VAX BASIC graphics statements allow you to optionally specify this number for each device. Statements with an optional identification clause include the control statements, many input statements, many ASK statements, and the statements for device transformations.

When you do not specify the identification clause, the statement affects only the default device (device #0). When you do specify this identification clause, each statement's effect is limited only to the device identified any other devices identified in the application are not affected. For example, the following SET statement changes the VALUE echo area only on device #1.

[1] Device identification numbers in OPEN . . . FOR GRAPHICS statements are not related to the channel numbers specified in OPEN statements. Channel numbers and device identification numbers are independent of each other.

```
OPEN "TV38" FOR GRAPHICS AS DEVICE #1
    .
    .
    .
SET VALUE ECHO AREA #1 : devx_min,devx_max,devy_min,devy_max
```

You can specify #0 in statements that include the optional device identification clause. However, you can include other devices in statements only if the device has already been explicitly opened and identified. When a statement specifies a device that has not been identified in an OPEN . . . FOR GRAPHICS statement, VAX BASIC signals an error. Remember that when you do not identify a device, VAX BASIC uses the default identification of 0.

The optional identification clause allows you to specify which device VAX BASIC should request and accept input from. The following example sets up a different menu for two devices:

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG work_choice, fun_choice
OPEN "office_term" FOR GRAPHICS AS DEVICE #1
OPEN "home_term" FOR GRAPHICS AS DEVICE #2
!+
!Set initial choice lists for each terminal with
!the device identification clause as first optional clause
!-
SET INITIAL CHOICE  #1 , LIST ("Program"     &
                              ,"Write"       &
                              ,"Review"      &
                              ,"Edit"        &
                              ,"Quit") : 5

SET INITIAL CHOICE #2 , LIST ("Play more"    &
                             ,"Quit") : 1
!+
!Request input from device #1
!-
LOCATE CHOICE #1 : work_choice
!+
!Request input from device #2
!-
LOCATE CHOICE #2 : fun_choice
    .
    .
    .
```

When several optional clauses are included in a statement, the optional device identification clause must be the first clause listed in the statement. The following statement includes a device identification clause followed by an optional COUNT clause. The statement accepts points from a user at the device identified as #2 and stores the number of points entered in the variable *How_many*.

```
MAT GET POINTS #2 , COUNT How_many : x_coords, y_coords
```

For syntax details of particular statements, see the reference section of thi manual.

8.1.2 Supported Device Types

Each hardware device supported by VAX GKS has an assigned number that indicates the type of device you are using and whether the device is to be used for input, output, or both. Device types that are supported by VAX GKS are listed with the OPEN . . . FOR GRAPHICS statement in th reference section.

When you use the OPEN . . . FOR GRAPHICS statement, you can optionally specify a device type. For example, the following statement opens "RT30", a system name for a color VT240, device type 13:

```
OPEN "RT30" FOR GRAPHICS AS DEVICE #2 , TYPE 13
```

When you do not specify a device type, VAX BASIC searches for an assignment to the logical name GKS$WSTYPE. If you have made no assignment, VAX BASIC searches for a system assignment. If the search fails, VAX BASIC uses the default device type of 14, which is a monochrome VT240. To find out if a system-wide assignment has been made, issue the following DCL command:

```
$ SHOW LOGICAL GKS$WSTYPE
```

If no translation is available for GKS$WSTYPE, or if you need to overrid the system assignment, you can assign the device type for your own device to GKS$WSTYPE in your startup procedures. For instance, in the login command procedure for a graphics session on a VT240 color terminal, include the following DCL command:

```
$ ASSIGN 13 GKS$WSTYPE
```

If you have already assigned your current device type with this comman and do not specify another device type in your program, then VAX BASI uses this assigned value by default and output is displayed, in this case, i the format for a VT240 color monitor.

You can have multiple devices of the default device type or any assigned device type. You should specify the device type in each OPEN . . . FOR GRAPHICS statement only if you cannot assign a device type number to GKS$WSTYPE, or when you use more than one device type. When you specify a device type, your program is no longer portable to another device.

The system name for the device must refer to an actual device of the type you specify. In the following example, the device RT247: is opened as device type 13, a VT240 with the color option. The system name RT247: refers to an actual device that is a color VT240.

```
OPEN "RT247:" FOR GRAPHICS AS DEVICE #2 , TYPE 13
```

Note that your system configuration may require an alternative method of specifying devices such as printers. For instance, to specify a queued printer on a VAXcluster, you may have to include the name of the node in the quoted string.

You cannot open a device with an incorrect device type. For instance, you cannot open a video display unit as a printer. The following statement opens an LA34 printer, device type 31, with a device identification of 1. The quoted string *my_printer* must be translated to the actual LA34 you are opening.

```
OPEN "my_printer" FOR GRAPHICS AS DEVICE #1, TYPE 31
```

Output is displayed on all active devices during program execution. The following example opens a VT240 and an LA50; the picture *ellipse* is displayed on the VT240 screen and printed on the LA50.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE ellipse
OPEN "my_printer" FOR GRAPHICS AS DEVICE #1, TYPE 32
OPEN "VTA83" FOR GRAPHICS AS DEVICE #2, TYPE 13

DRAW ellipse
    .
    .
    .
```

.1.3 Controlling Devices

Because output is displayed on all active output devices, VAX BASIC provides the capability to explicitly inhibit the display of output on specified devices. For instance, you could open three devices for interactive graphics, yet want the output displayed on only two of them. To inhibit a display on a device, use the DEACTIVATE DEVICE statement; to activate the device again, use the ACTIVATE DEVICE statement. The device identification number is required with each of these statements.

When VAX BASIC opens a device with the OPEN . . . FOR GRAPHICS statement, the device is also automatically activated. When no devices are explicitly opened and graphics statements are executed, the default device is automatically opened and activated. After all devices are opene and identified, you can inhibit the display going to one device with the DEACTIVATE DEVICE statement. For example:

```
DEACTIVATE DEVICE #5
```

When you want to display an image on this device later, use the ACTIVATE DEVICE statement.

```
ACTIVATE DEVICE #5
```

You can explicitly deactivate the default device (#0); however, if no devices are active and a graphics output statement is executed, VAX BASIC reactivates the default device.

The CLEAR statement allows you to clear the display on a particular device. VAX BASIC clears a device automatically when it is first opened. When no device identification is included in the CLEAR statement, only the display on the default device is cleared.

The following example uses these three statements to control the display of pictures when several devices are active. Logical names are used to represent the actual terminals.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE chocolate_candy,salad,balanced_meal

OPEN "cafe_term" FOR GRAPHICS AS DEVICE #1, TYPE 13
OPEN "Marys_term" FOR GRAPHICS AS DEVICE #2, TYPE 12
OPEN "Evs_term" FOR GRAPHICS AS DEVICE #3, TYPE 12
    .
    .
    .
```

```
DEACTIVATE DEVICE #3
DRAW chocolate_candy
ACTIVATE DEVICE #3
DEACTIVATE DEVICE #2
DRAW salad
ACTIVATE DEVICE #2
CLEAR #1
CLEAR #2
CLEAR #3
DRAW balanced_meal
SLEEP 10%
END
```

Chocolate_candy is not displayed on the device *Evs_term,* and *salad* is not displayed on *Marys_term,* while *balanced_meal* is displayed on all three devices.

The RESTORE GRAPHICS statement allows you to restore all the graphics defaults and attributes to the values at the start of program execution. This statement does not accept a device identification; when you use RESTORE GRAPHICS, all open devices are closed and all graphics attributes and defaults are restored to the initial state at the start of program execution.

You can explicitly close a device with the CLOSE DEVICE statement. This statement is optional; when a program is terminated, VAX BASIC automatically closes all open devices. However, it is good programming practice to explicitly close any devices that you explicitly open with an OPEN . . . FOR GRAPHICS statement. You can also use the CLOSE DEVICE statement as a safeguard in error handlers, as shown in the following example. This program displays the picture *circle* on devices #1 and #2. If an error is encountered while device #2 is being opened, device #1 is closed and *circle* is displayed on the default device.

Example

```
PROGRAM open_up

OPTION TYPE = EXPLICIT
EXTERNAL PICTURE circle
DECLARE LONG times

OPEN "RTA1" FOR GRAPHICS AS DEVICE #1, TYPE 12
WHEN ERROR IN
     OPEN "exper_term" FOR GRAPHICS AS DEVICE #2, TYPE 12
USE
     GRAPH TEXT AT 0.5,0.5: "Problems with experimental terminal"
     CLOSE DEVICE #1
     EXIT HANDLER
END WHEN

DRAW circle WITH SHEAR(10 * times) FOR times = 1% TO 5%
END PROGRAM
```

8.2 Metafiles

When a program creates a graphics image, you can store this image in a file known as a *metafile*. Metafiles provide you with a convenient methoc of storing and reproducing graphics images. They allow you to recreate an image without repeatedly processing the data. Metafiles are especially useful for:

- Recreating complex images that are used frequently
- Recreating images from user-supplied input that could not easily be reentered

8.2.1 Creating Metafiles

To send an image to a metafile, your application program must open a metafile as an alternative device. The following OPEN . . . FOR GRAPHICS statement opens the file *grad_swans.pic* as device #1 with the device type 2, an output metafile.

```
device_type = 2
OPEN "grad_swans.pic" FOR GRAPHICS AS DEVICE #1, TYPE device_type
```

Graphics output statements executed after this OPEN . . . FOR GRAPHICS statement are now directed to device #1, the metafile *grad_swans.pic*. You cannot open an existing metafile as device type 2; VAX BASIC can open only a new file.

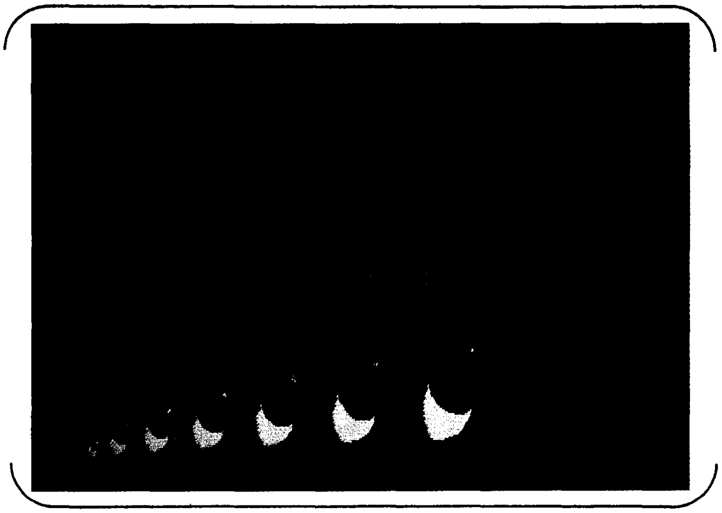
The following example opens two devices: a VAXstation II and a metafilε The program uses the picture *swan* defined in Chapter 6. It displays the image on device #2 (a VAXstation) and stores the image data in the file *grad_swans.pic*, device #1. If no directory is specified, the metafile is created in your current directory.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE swan
DECLARE SINGLE grad_matrix(4,4)
DECLARE LONG counter

OPEN "grad_swans.pic" FOR GRAPHICS AS DEVICE #1, TYPE 2
OPEN "VTA72" FOR GRAPHICS AS DEVICE #2 , TYPE 41
```

```
SET WINDOW , TRAN 1 : 0,100,0,100
FOR counter = 1% TO 7%
  MAT grad_matrix = SHIFT((counter * counter) + (counter * 30),0)     &
                    * SCALE(1/25 * counter)
  DRAW swan WITH grad_matrix
NEXT counter
SET TEXT HEIGHT 3
GRAPH TEXT AT 5,40 :"Seven Swans A-Graduating"

CLOSE DEVICE #1
CLOSE DEVICE #2
END
```

Output



The image created by this DRAW statement is now stored in the metafile *grad_swans.pic* and displayed on the VAXstation II. The metafile *grad_swans.pic* can now be displayed on any open output device, as described in the following section.

8.2.2 Displaying the Contents of a Metafile

Metafiles contain data records identifying the VAX GKS functions requir for an output display. Once a valid VAX GKS metafile has been created you can display the contents of the metafile with the GRAPH METAFIL statement. The contents of the metafile are displayed on all active devic The GRAPH METAFILE statement requires the file specification of a val metafile or a logical name. For example:

```
GRAPH METAFILE "[HEINES.GRAPHICS]grad_swans.pic"
```

Images contained in metafiles are displayed with the window and view- port specified when the metafile was created, unless you set alternatives before the GRAPH METAFILE statement is executed. To display the metafile created by the previous example, you must include the GRAPH METAFILE statement in a program. This example opens an output devi and displays the metafile. If no output device is explicitly opened for th display, VAX BASIC displays the metafile on the default device.

Example

```
OPEN "office_term" FOR GRAPHICS AS DEVICE #1, TYPE 13
SET WINDOW , TRAN 1 : 0,100,0,100
GRAPH METAFILE "[HEINES.GRAPHICS]grad_swans.pic"
END
```

The following program *Childs_play* stores a user's drawings in metafiles and displays them when requested. The drawings can be saved from one session to another. The example assumes that a user knows whethe any previous metafiles have been created, and that the user knows the names of the drawings. Only the current drawing is held in memory. Two devices are used: an output metafile, and the default device type fc the user's terminal.

Example

```
PROGRAM childs_play

  OPTION TYPE = EXPLICIT
  DECLARE LONG to_do,how_many,                                &
          SINGLE x_array(150), y_array(150)
  EXTERNAL SUB display,                                       &
               save_it(SINGLE DIM (), SINGLE DIM (), LONG)
  OPEN "" FOR GRAPHICS AS DEVICE #1
  UNTIL 0%
      !+
      !Main menu
      !-
      SET INITIAL CHOICE , LIST   ("Draw a new picture"        &
                                  ,"Save the current picture"  &
                                  ,"View an old picture"       &
                                  ,"Quit this program")        &
                                  : 1
      LOCATE CHOICE #1 : to_do

      SELECT to_do
          CASE = 1
                 CLEAR
                 MAT LOCATE POINTS , #1                  &
                                   , COUNT how_many      &
                                   : x_array, y_array
                 IF how_many > 151
                    THEN how_many = 151
                 END IF
          CASE = 2
                 CALL save_it(x_array(),y_array(),how_many)
          CASE = 3
                 CALL display
          CASE = 4
                 EXIT PROGRAM
      END SELECT

  NEXT
END PROGRAM

SUB display

   OPTION TYPE = EXPLICIT
   DECLARE STRING CONSTANT present =                      &
                          "Which drawing? ( <CR>  = back to menu) "
   DECLARE STRING meta_file
   SET INITIAL STRING #1 : present

   UNTIL 0%
     WHEN ERROR IN
       LOCATE STRING #1 , INITIAL present : meta_file
       IF meta_file = present
          THEN EXIT SUB
          ELSE
              meta_file = MID$(meta_file,38,46)
              GRAPH METAFILE meta_file + ".met"
              SLEEP 15%
              CLEAR
       END IF
      USE
        IF ERR = 5%
           THEN GRAPH TEXT AT 0.05,0.95 : "That drawing does not exist,"
                GRAPH TEXT AT 0.05,0.85 : "please try another."
                RETRY
        END IF
      END WHEN
   NEXT
END SUB

SUB save_it(SINGLE x_array(),y_array (),how_many)
  OPTION TYPE = EXPLICIT
  DECLARE STRING meta_file
  LOCATE STRING , INITIAL "Give it a name> ": meta_file
  meta_file = MID$(meta_file,17,46)

  !+
  !Inhibit display from user's terminal
  !-
  DEACTIVATE DEVICE #1
  OPEN (meta_file + ".met" FOR GRAPHICS AS DEVICE #2, TYPE 2
  MAT PLOT LINES , COUNT how_many : x_array, y_array

  !+
  !Reactivate user's terminal
  !
  ACTIVATE DEVICE #1
  !+
  !Close metafile
  !-
  CLOSE DEVICE #2
END SUB
```

8.3 Determining Device Capabilities

You may need to retrieve information about a device when your applica
tion is being used on an unknown device; for instance, an application m
use the VAX BASIC default values and have no control over the hardwa
device on which a user might run the application. To retrieve informati
about hardware devices in use, VAX BASIC provides two ASK statemen

- ASK DEVICE TYPE
- ASK DEVICE SIZE

The ASK DEVICE TYPE statement retrieves the device type for a devic
previously specified in an OPEN . . . FOR GRAPHICS statement, or a
device implicitly opened by VAX BASIC. For example, the following AS
statement retrieves information about the device identified as device #1.

```
ASK DEVICE TYPE #1 : device_type
```

Once you know the device type, you can branch to alternative routines in your program. The following example requests information about an unknown device. The application proceeds depending on the values retrieved by the ASK statements. VAX BASIC assigns a value for the device type to the variable *dev_type* in the ASK DEVICE TYPE statement.

Example

```
OPTION TYPE = EXPLICIT

EXTERNAL PICTURE sine_curve
EXTERNAL SUB VS_start_up,color_rtn
DECLARE LONG dev_type
OPEN "user_term" FOR GRAPHICS AS DEVICE #1

ASK DEVICE TYPE #1 : dev_type

SELECT dev_type
       CASE = 41
            CALL VS_start_up
       CASE = 13
            CALL color_rtn
       CASE ELSE
            DRAW sine_curve
            CLOSE DEVICE #1
END SELECT
    .
    .
    .
```

The ASK DEVICE SIZE statement retrieves the size of the available display surface of the device in use. For instance, the following ASK statement requests the size of device opened as 41, a VAXstation II:

Example

```
DECLARE SINGLE horizontal_size,vertical_size,                      &
        STRING what_measure
OPEN "my_term" FOR GRAPHICS AS #1 , TYPE 41
ASK DEVICE SIZE #1 : horizontal_size, vertical_size, what_measure
    .
    .
    .
```

For VAXstations, the value assigned to *what_measure* is "METERS". When the units of measure are not meters, the string assigned is "OTHER". The measurement of the available display surface in the horizontal direction is assigned to *horizontal_size*; the measurement in the vertical direction is assigned to *vertical_size*. You can use these assigned values to change echo area boundaries or device viewport boundaries as shown in Section 8.4.

Additional information can be retrieved about specific devices with the following statements. Each of these statements can include an optional device identification.

- ASK MAX LINE SIZE
- ASK MAX POINT SIZE
- ASK MAX COLOR
- ASK COLOR MIX
- ASK...ECHO AREA

These statements each retrieve device-specific information that you ma) not be able to predict before run time. When specified, the device ident cation clause must be the first optional clause in the statement. For moı details about each of these statements, see Chapters 3, 4, and 7 as well the reference material in Chapter 9.

Appendix B provides device-specific information, such as device coordinates, for VAXstations and VT125 and VT240 terminals.

8.4 Device Transformations

In addition to the normalization transformations described in Chapter 5 you can also transform an image from NDC space to your device. This is known as a *device transformation.* Device transformations allow you t make use of all the available display space on an output device. Howev device transformations are dependent on the size of a particular device, unlike normalization transformations, which are device independent.

To recap, the normalization transformation defines the projection of an image in the world window onto a viewport in NDC space. Normalizati transformations include:

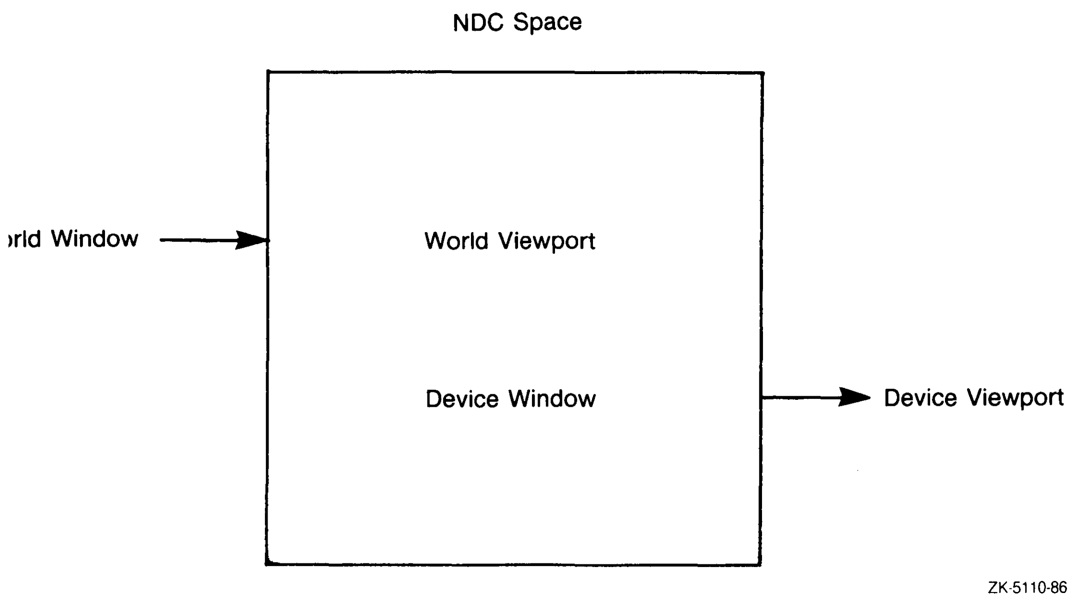
- Setting world window boundaries with the SET WINDOW statemen
- Setting world (NDC) viewport boundaries with the SET VIEWPOR1 statement
- Establishing the current transformation with the SET TRANSFORM/ statement
- Clipping the graphics image within the viewport boundaries with th SET CLIP statement

A device transformation defines the projection of a portion of NDC space onto a device viewport. Device transformations include:

- Selecting a portion of NDC space as the device window with the SET DEVICE WINDOW statement
- Selecting a portion of the device as a device viewport with the SET DEVICE VIEWPORT statement

Figure 8–1 illustrates the progression of both of these kinds of transformations.

ıure 8–1: Normalization and Device Transformations



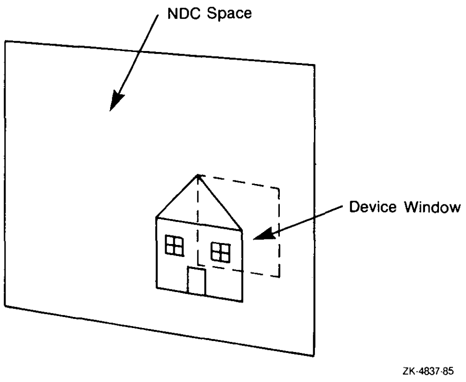
The device window is the portion of NDC space to be projected onto your screen. The default device window includes all of NDC space with left, right, bottom, and top boundaries of 0,1,0,1 respectively. You can specify that only a portion of NDC space is projected onto the screen with the SET DEVICE WINDOW statement. You cannot select a window larger than the NDC space limits. A complementary ASK DEVICE WINDOW statement allows you to retrieve the current device window boundaries.

The following example sets the device window to 0.65,0.85,0.3,0.5. Figure 8–2 illustrates the device window specified.

Example

```
OPEN "VTA72" FOR GRAPHICS AS DEVICE #1, TYPE 12
SET DEVICE WINDOW #1 : 0.65,0.85,0.3,0.5
    .
    .
    .
```

Figure 8–2: The Device Window in NDC Space



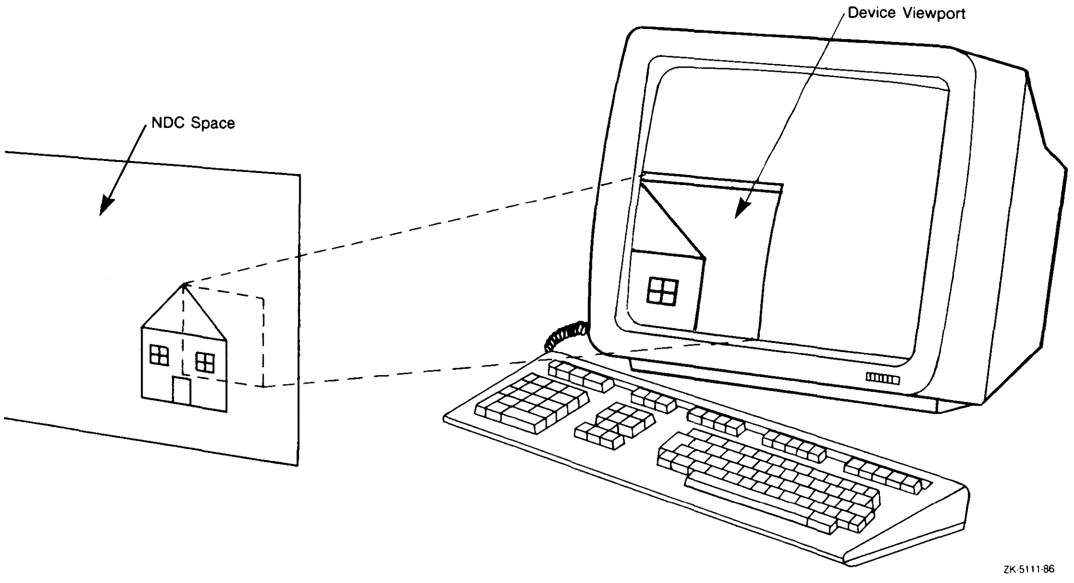
The image defined in the device window of NDC space is projected ont the device viewport, which is a portion of your screen. The default size and shape of the device viewport is device dependent; however, it is commonly the largest square area your device can accommodate.

You can change the boundaries of the device viewport with the SET DEVICE VIEWPORT statement. The following example identifies the device as #1, retrieves the device coordinates with the ASK DEVICE SIZE statement, and uses these coordinates to set a new device viewpor Figure 8–3 illustrates the projection of the previous device window onto square device viewport.

Example

```
DECLARE SINGLE screen_height,screen_width
OPEN "VTA101" FOR GRAPHICS AS DEVICE #1
SET DEVICE WINDOW #1 : 0.65,0.85,0.3,0.5
ASK DEVICE SIZE screen_width, screen_height
SET DEVICE VIEWPORT #1 : 0, screen_width/2,          &
                         0, screen_width/2
    .
    .
    .
```

gure 8–3: **The Device Window Is Projected onto the Device Viewport**



The actual boundaries of the device viewport are dependent on the device. For instance, VAXstations can have multiple viewing windows, while the full display area on VT125 and VT240 screens is rectangular. The following example shows how to extend the default device viewport to the available display surface of VT125 and VT240 terminals. If you do not know the dimensions of the particular output device you are working with, use the ASK DEVICE SIZE statement to retrieve the appropriate values. You can also use the ASK DEVICE VIEWPORT statement to retrieve the current boundaries of the device viewport. Once you know the exact dimensions of your screen, you can change the boundaries of th device viewport with the SET DEVICE VIEWPORT statement.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE horiz_size, vert_size,           &
        STRING what_measure

OPEN "VTA247" FOR GRAPHICS AS DEVICE #1
ASK DEVICE SIZE #1, horiz_size, vert_size, what_measure
!+
!Use these values to set the device viewport
!to the full extent of your screen
!-
SET DEVICE VIEWPORT #1 : 0, horiz_size, 0, vert_size
    .
    .
    .
```

Note that both the default device window and the default device viewpoı are square in shape. Figure 8–3 illustrates a square device window that is projected onto a square device viewport. When the aspect ratio is not 1:1, VAX BASIC overrides your specifications for a device viewport and uses the largest possible device viewport that matches the shape of the device window. To ensure that the aspect ratio is 1:1, you should set a rectangular device window in direct proportion to a rectangular device viewport.

You cannot alter the clipping status in terms of the output device. If a graphics object extends beyond the actual boundaries of the device window or viewport you specify, the excess of the image is not displayed

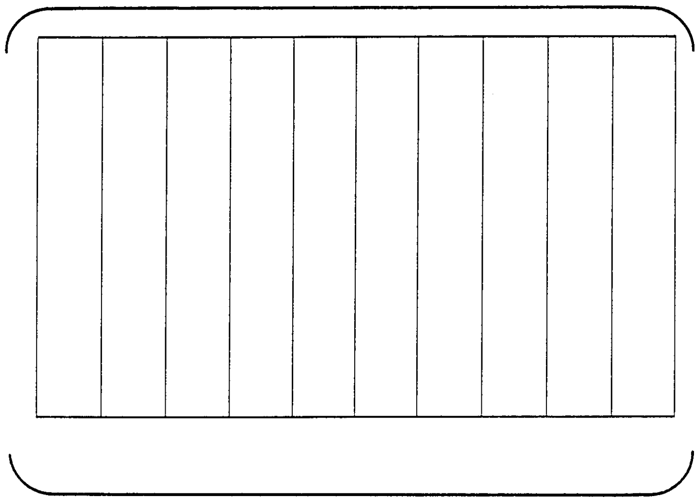
Note that device transformation information is ignored in metafiles and metafile output is displayed using the current device window and viewport.

The following subprogram uses the whole screen as the device viewport. You can declare and call this subprogram from a graphics main program, as shown here.

Example

```
SUB Full_screen(LONG trans)
 OPTION TYPE = EXPLICIT
 DECLARE STRING temp
 DECLARE SINGLE aspect_ratio, dcx_max, dcy_max
 SET TRANSFORMATION trans
 SET CLIP "OFF"

 ASK DEVICE SIZE dcx_max, dcy_max
 aspect_ratio = dcy_max/dcx_max
 !+
 !Same ratio as device
 !-
 SET VIEWPORT 0,1,0,aspect_ratio
 !+
 !Full screen
 !-
 SET DEVICE VIEWPORT 0,dcx_max,0,dcy_max
 SET DEVICE WINDOW 0,1,0,aspect_ratio
END SUB

PROGRAM Demo
 EXTERNAL SUB Full_screen(LONG)
 DECLARE SINGLE counter

 CALL Full_screen(1)
 GRAPH LINES 0,1; 1,1; 1,0; 0,0; 0,1
 GRAPH LINES counter,0.0;counter,1.0 FOR counter = 0 TO 1 STEP 0.5
END PROGRAM
```

Output



Although complex manipulations with device viewports are possible, it is preferable to manipulate images in NDC space. Rather than divide the device viewport into various portions, it is less cumbersome to build a composite or complex display in NDC space using normalization transfor mations. Once such a composite display is created in NDC space, all of NDC space can be mapped to the default device viewport. It is more reliable (and easier) to use normalization transformations rather than device transformations to manipulate composite images because normalization transformations are device independent.

8.5 Summary

This chapter has discussed how to:

- Specify different devices in graphics control and input statements, such as CLOSE DEVICE and LOCATE CHOICE
- Use metafiles to store graphics output and display the files with the GRAPH METAFILE statement
- Determine device capabilities with the ASK DEVICE TYPE and ASK DEVICE SIZE statements, as well as other statements
- Define device transformations with the SET DEVICE WINDOW and SET DEVICE VIEWPORT statements

ꞩEFERENCE SECTION

Chapter 9

VAX BASIC Graphics Statements

This chapter provides reference material on each VAX BASIC graphics statement. The descriptions are arranged in alphabetical order and include the following sections:

Overview	An overview of what the statement does.
Format	The required syntax for the statement.
Syntax Rules	Any rules governing the use of parameters, separators, or other syntax items.
Remarks	Explanatory remarks concerning the effect of the statement on program execution and any restrictions governing its use.
Example	One or more examples of the statement in a program. Where appropriate, sample output is shown.

Explanation of Syntax Diagrams

The following tables explain the conventions used in the syntax diagrams in this chapter. The syntax diagrams consist of VAX BASIC keywords, mnemonics, and punctuation symbols.

The following symbols are used in the syntax diagrams.

Symbol	Meaning
UPPERCASE	Uppercase words are VAX BASIC keywords that must be coded exactly as shown.
lowercase	Lowercase words are elements that must be replaced by an appropriate user-supplied value.
[ ]	Brackets enclose an optional portion of a format. Brackets around vertically stacked items indicate that you can select one of the enclosed items. You must include all punctuation as it appears in the brackets.
{ }	Braces enclose a mandatory portion of a format. Braces around vertically stacked items indicate that you must choose one of the enclosed items. You must include all punctuation as it appears in the braces.
. . .	An ellipsis indicates that the immediately preceding item can be repeated. An ellipsis following a format unit enclosed in brackets or braces means that you can repeat the entire unit. If repeated items or format units must be separated by commas, the ellipsis is preceded by a comma (, . . .).

The following mnemonics are used in the syntax diagrams:

Mnemonic	Meaning
angle	An angle in radians or degrees
array	An array; syntax rules specify whether the bounds or dimensions can be specified
cond-exp	Conditional expression; used to indicate that an expression can be either logical or relational
const	A constant value
data-type	A data type keyword
def	Specific to a DEF function
dev-id	An identification number for a particular graphics device; always preceded by a number sign (#)
dev-type	The number associated with the particular type of hardware device (see Table 9–1)
exp	An expression

Mnemonic	Meaning
file-spec	A file specification
int	An integer
int-exp	An expression that represents an integer value
int-var	A variable that contains an integer value
label	An alphanumeric statement label
line	A statement line; may or may not be numbered
line-num	A statement line number
matrix	A two-dimensional array
name	A name or identifier; indicates the declaration of a name or the name of a VAX BASIC structure, such as a PICTURE subprogram
num	A numeric value
param-list	A parameter list, such as for a PICTURE subprogram
pass-mech	A valid VAX BASIC passing mechanism
pic-name	The name of a PICTURE subprogram
real	A floating-point value
real-exp	A floating-point expression
real-var	A floating-point variable
str	A character string
str-exp	An expression that represents a character string
str-var	A variable that contains a character string
target	The target point of a branch statement; either a line number or a label
tran-term	A reserved keyword used for a transformation function
unsubs-var	Unsubscripted variable; used to indicate a simple variable, as opposed to an array element
var	A variable
x-array	An array of values for x-coordinates
x-coord	A value for the x-coordinate of a point
y-array	An array of values for y-coordinates
y-coord	A value for the y-coordinate of a point

ACTIVATE DEVICE

The ACTIVATE DEVICE statement allows you to activate an output device that has been explicitly deactivated. Subsequent graphics output is displayed on the specified device.

Format

ACTIVATE DEVICE *#dev-id*

Syntax Rules

You must supply the number sign (#) with *dev-id*.

Remarks

1. *Dev-id* specifies the device that should be activated. Unless you specify the default device, the device specified must have been identified with an OPEN . . . FOR GRAPHICS statement.
2. A device must be open and activated for graphics output to be displayed on that device. The OPEN . . . FOR GRAPHICS statement implicitly activates an output device.
3. After a device has been deactivated, graphics output is not displayed on that device until an ACTIVATE DEVICE statement is executed.
4. If no devices are open and activated when an output statement is executed, VAX BASIC implicitly opens and activates the default device (device #0).
5. See also the DEACTIVATE DEVICE statement.

Example

```
EXTERNAL PICTURE Basic_crowd,Basic_party_crowd
OPEN "VTA247" FOR GRAPHICS AS DEVICE #2
OPEN "RT1" FOR GRAPHICS AS DEVICE #3
   .
   .
   .
!+
!Send output only to device #3
!-
DEACTIVATE DEVICE #2
DRAW Basic_crowd
!+
!Send output to device #2 and #3
!-
ACTIVATE DEVICE #2
DRAW Basic_party_crowd
```

ASK AREA COLOR

The ASK AREA COLOR statement allows you to retrieve the current color index for areas.

Format

ASK AREA COLOR *int-var*

Syntax Rules

None.

Remarks

1. The range of valid color indices is device dependent. Results are unpredictable if you use the value assigned to *int-var* in a subsequent SET AREA COLOR statement and the index is undefined for any activated device.
2. On VT125 and VT240 terminals with the color option, the default value assigned to *int-var* represents one of the following colors:

Index	Color
0	Black
1	Green
2	Red
3	Blue

3. See also the SET AREA COLOR, SET/ASK COLOR MIX, and ASK MAX COLOR statements.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG color_var
ASK AREA COLOR color_var
SET AREA COLOR color_var + 1
GRAPH AREA 0.0,0.0; 1.0,1.0; 1.0,0.0
END
```

ASK AREA STYLE

The ASK AREA STYLE statement retrieves the value for the current area style.

Format

ASK AREA STYLE *str-var*

Syntax Rules

None.

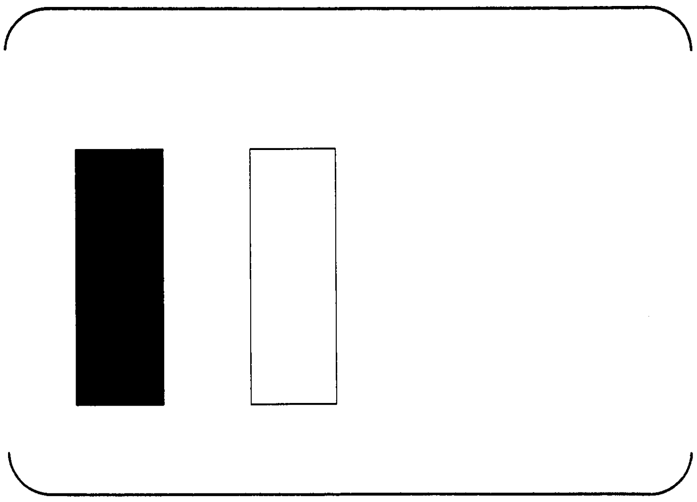
Remarks

1. The value assigned to *str-var* is one of four possible values:
   - HOLLOW
   - SOLID
   - PATTERN
   - HATCH
2. The default area style is solid.
3. If the area style is HATCH or PATTERN, various index styles are defined in the area style index. See the SET/ASK AREA STYLE INDEX statements.

xample

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE strip
DECLARE STRING which_style
ASK AREA STYLE which_style
DRAW strip
IF which_style = "SOLID"
   THEN SET AREA STYLE "HOLLOW"
END IF
DRAW strip WITH SHIFT(0.4,0)
END

PICTURE strip
DIM SINGLE x_strip(3),y_strip(3)
  x_strip(0) = 0.1
  x_strip(1) = 0.3
  x_strip(2) = 0.3
  x_strip(3) = 0.1
  y_strip(0) = 0.7
  y_strip(1) = 0.7
  y_strip(2) = 0.1
  y_strip(3) = 0.1
  MAT PLOT AREA x_strip, y_strip
END PICTURE
```

Output



ZK-4948-86

ASK AREA STYLE INDEX

The ASK AREA STYLE INDEX statement retrieves the value for the current area style index. The index value specifies one of the various hatch or pattern styles.

Format

ASK AREA STYLE INDEX *int-var*

Syntax Rules

None.

Remarks

1. The area style index specifies one of the various hatch or pattern styles. There is no index value for hollow or solid area styles.
2. Area index values are device dependent. If you use the value assigne to *int-var* in a subsequent SET AREA STYLE INDEX statement and th index specified is not defined for the device, results are unpredictable when an area is subsequently displayed. Appendix B lists possible indices for VAXstations and VT125 and VT240 terminals.
3. Examples of possible indices are shown with the SET AREA STYLE INDEX statement and also in Chapter 3 of this manual.

Example

```
DECLARE LONG which_ind
ASK AREA STYLE INDEX which_ind
IF which_ind > 32%
   THEN SET AREA STYLE INDEX (which_ind - 32%)
END IF
    .
    .
    .
```

ASK CHOICE ECHO AREA

The ASK CHOICE ECHO AREA statement allows you to retrieve the boundaries of the current CHOICE echo area. The boundaries are assigned in device coordinates.

Format

```
                         ┌                        ┐
ASK CHOICE ECHO AREA     │ #dev-id        ┌   ┐   │
                         │ , UNIT int-exp │ : │   │
                         │                └   ┘   │
                         └                        ┘
                   real-var1, real-var2, real-var3, real-var4
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. *Real-var1* must be preceded by a colon if one or more optional clauses are included.
3. *Real-var1, real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

Remarks

1. The echo area is the portion of your screen where the prompt appears and where input can be supplied by a user. The default boundaries are device dependent.
2. VAX BASIC assigns the boundaries for this area in device coordinates

3. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
4. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the selection from a menu can be indicated with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
5. See also the SET CHOICE ECHO AREA statement.
6. Note that you cannot retrieve or change the boundaries of the echo area for POINT or MULTIPOINT input. These echo areas coincide with the default for the device.

xample

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE xmin,xmax,ymin,ymax,                                 &
               new_xmin,new_xmax,new_ymin,new_ymax,                 &
        LONG which
ASK CHOICE ECHO AREA xmin, xmax, ymin, ymax
!+
!Set new echo boundaries
!-
new_xmin = 0
new_xmax = xmax
new_ymin = ymin
new_ymax = ymax
SET CHOICE ECHO AREA new_xmin, new_xmax, new_ymin, new_ymax
SET INITIAL CHOICE , LIST ("Richer"         &
                          ,"Poorer"         &
                          ,"Better"         &
                          ,"Worse")         &
                          : 3
LOCATE CHOICE which
   .
   .
   .
```

ASK CLIP

The ASK CLIP statement allows you to determine whether clipping is currently on or off.

Format

ASK CLIP *str-var*

Syntax Rules

None.

Remarks

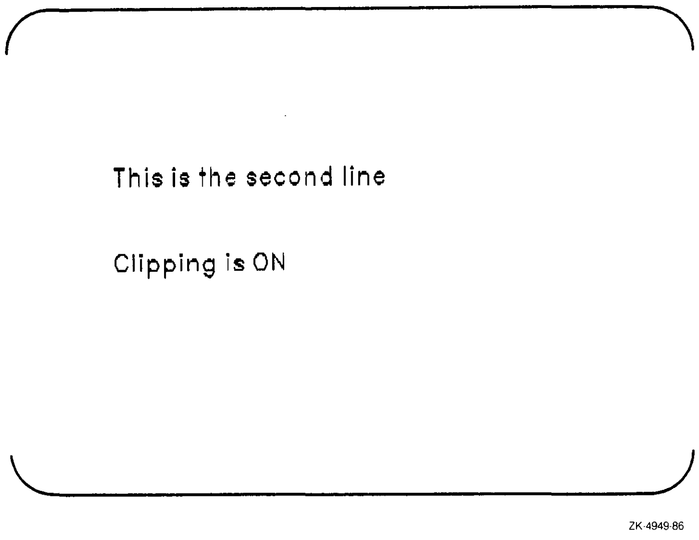
1. The value assigned to *str-var* is either "ON" or "OFF".
2. Clipping is enabled at the start of program execution.
3. When clipping is enabled with the SET CLIP "ON" statement, graphic images with world coordinate values exceeding the limits of the worl window are not displayed.
4. When clipping is disabled with the SET CLIP "OFF" statement, image with world coordinate values that exceed the limits of the world window are displayed, provided that the points are also within the device window.
5. When clipping is enabled, STRING precision clips a text string at the world viewport boundary. No text string starts beyond the world viewport boundary.
6. When clipping is enabled, CHAR precision clips each text character at the world viewport boundary. No character extends beyond the world viewport boundary. Some devices display part of a character that spans the world viewport boundary.

7. When clipping is enabled, STROKE precision clips text precisely at the world viewport boundary. For example, if only half a character falls inside the world viewport boundary, only this half is displayed on the screen.

xample

```
DECLARE STRING clipping
SET WINDOW , TRAN 2 : 0,1,0,0.5
SET TEXT HEIGHT 0.03
ASK CLIP clipping
!+
!Graph text - the top line is beyond the boundaries
!of the world window for TRAN 2
!-
GRAPH TEXT AT 0.2,0.7 : "This is the top line"
GRAPH TEXT AT 0.2,0.3 : "This is the second line"
GRAPH TEXT AT 0.2,0.2 : "Clipping is " + clipping
```

Output



ASK COLOR MIX

The ASK COLOR MIX statement retrieves values for the intensities of rec green, and blue associated with the current color index.

Format

ASK COLOR MIX *[#dev-id]*, **INDEX** *int-exp* : *real-var1*, *real-var2*, *real-var3*

Syntax Rules

1. When specified, *dev-id* must be preceded by the number sign (#).
2. You specify the color index with *int-exp*.
3. *Real-var1* retrieves the intensity associated with red.
4. *Real-var2* retrieves the intensity associated with green.
5. *Real-var3* retrieves the intensity associated with blue.

Remarks

1. The optional *dev-id* identifies the device for which you want informa- tion. If no identification clause is included, VAX BASIC uses a defaul identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. Values for the intensities of the colors red, green, and blue are greate than or equal to 0.0 and less than or equal to 1.0.

3. The default intensity values for color indices 1, 2, and 3 are as follows:

Color Index	Color	Red Intensity	Green Intensity	Blue Intensity
0	Black	0.0	0.0	0.0
1	Green	0.0	1.0	0.0
2	Red	1.0	0.0	0.0
3	Blue	0.0	0.0	1.0

4. See the SET COLOR MIX statement for more information. Possible values for the intensities are listed in Appendix B.
5. The number of color indices is device dependent. If you use the values retrieved with this statement to set colors on a device that does not support the same indices, results are unpredictable.

xample

```
DECLARE SINGLE red, green, blue
OPEN "color_term" FOR GRAPHICS AS DEVICE #1
!+
!Ask for the 3 default intensities associated with INDEX 2
!-
ASK COLOR MIX #1 , INDEX 2 : red, green, blue
GRAPH TEXT AT 0.05,0.8 : "Red intensity = " + STR$(red)
GRAPH TEXT AT 0.05,0.7 : "Green intensity = " + STR$(green)
GRAPH TEXT AT 0.05,0.6 : "Blue intensity = " + STR$(blue)
```

ASK DEVICE SIZE

The ASK DEVICE SIZE statement lets you determine the actual size of th
available display surface on a particular device.

Format

ASK DEVICE SIZE *[#dev-id :] real-var1, real-var2 [, str-var]*

Syntax Rules

1. The optional *dev-id* must be preceded by a number sign (#) and followed by a colon (:).
2. The size in the horizontal direction is assigned to *real-var1*.
3. The size in the vertical direction is assigned to *real-var2*.
4. The unit of measure used is assigned to the optional *str-var*.

Remarks

1. The optional *dev-id* identifies the device for which you want informa tion. If no identification clause is included, VAX BASIC uses a defaul identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASI(as the default device.
2. The value assigned to the optional *str-var* is the unit of measure use(to interpret *real-var1* and *real-var2*. *Str-var* can be either "METERS" or "OTHER", indicating that the unit of measure is either meters or device coordinates.
3. See Chapter 8 for more examples.

xample

```
OPEN "VT101" FOR GRAPHICS AS DEVICE #1
DECLARE STRING measure
DECLARE SINGLE x_dev,y_dev

ASK DEVICE SIZE #1 : x_dev, y_dev
!+
!Use retrieved coords to set a new echo area
!-
SET CHOICE ECHO AREA #1 : 0,x_dev,0,y_dev
    .
    .
    .
END
```

ASK DEVICE TYPE

The ASK DEVICE TYPE statement retrieves the device type of the specified device.

Format

ASK DEVICE TYPE *[#dev-id :] int-var*

Syntax Rules

1. When specified, *dev-id* must be preceded by the number sign (#) and followed by a colon (:).
2. The value assigned to *int-var* is a valid supported device type.

Remarks

1. The optional *dev-id* identifies the device for which you want informa tion. If no identification clause is included, VAX BASIC uses a defaul identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASI(as the default device.
2. Valid device type values are shown in Table 9–1 with the OPEN . . . FOR GRAPHICS statement in this Chapter. This list is complete up to the print date on this manual. For the most up-to-date informatioı consult the VAX GKS documentation.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG what_type
OPEN "RT13" FOR GRAPHICS AS DEVICE #2
!+
!Ask for device type of opened device
!-
ASK DEVICE TYPE #2 : what_type
IF what_type = 41
   THEN CALL wkstation_proc
END IF
   .
   .
   .
```

ASK DEVICE VIEWPORT

The ASK DEVICE VIEWPORT statement retrieves the current values for the boundaries of the device viewport rectangle.

Format

ASK DEVICE VIEWPORT *[#dev-id :] real-var1, real-var2, real-var3, real-var4*

Syntax Rules

1. When specified, *dev-id* must be preceded by the number sign (#) and followed by a colon (:).
2. *Real-var1,real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

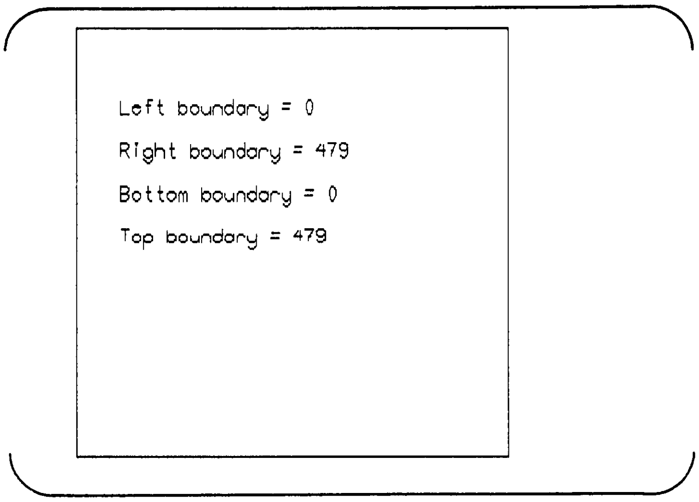
Remarks

1. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. A device viewport is the display area on an output device.
3. If no previous SET DEVICE VIEWPORT statement has been executed, the values assigned are the default viewport boundaries. If a SET DEVICE VIEWPORT statement has been executed, the assigned boundaries are equal to the boundaries last established in that statement unless VAX BASIC overrides those values. When you do not maintain an aspect ratio of 1:1, VAX BASIC overrides the viewport values with values that maintain the correct ratio.

:xample

```
OPTION TYPE = EXPLICIT
OPEN "VTA247" FOR GRAPHICS AS DEVICE #1
DECLARE SINGLE left_1,right_1,bottom,top
SET WINDOW 0,100,0,100
SET TEXT HEIGHT 4
ASK DEVICE VIEWPORT #1 : left_1, right_1, bottom, top
!+
!Draw the boundaries of the default viewport of the device
!-
GRAPH LINES 0,0; 100,0; 100,100; 0,100; 0,0
GRAPH TEXT AT 10,80 : "Left boundary = "  + STR$(left_1)
GRAPH TEXT AT 10,70 : "Right boundary = " + STR$(right_1)
GRAPH TEXT AT 10,60 : "Bottom boundary = " + STR$(bottom)
GRAPH TEXT AT 10,50 : "Top boundary = " + STR$(top)
END
```

Output



```
Left boundary = 0
Right boundary = 479
Bottom boundary = 0
Top boundary = 479
```

ZK-5516-86

ASK DEVICE WINDOW

The ASK DEVICE WINDOW statement allows you to retrieve the current values of the boundaries for the device window.

Format

ASK DEVICE WINDOW *[#dev-id :] real-var1, real-var2, real-var3, real-var4*

Syntax Rules

1. When specified, *dev-id* must be preceded by a number sign (#) and followed by a colon (:).
2. *Real-var1, real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

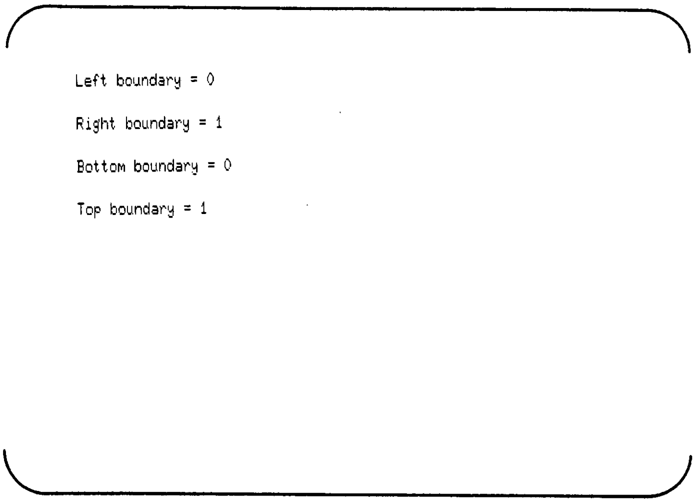
Remarks

1. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. A device window is the portion of NDC space you select for display.
3. The default device window boundaries specify the entire NDC space. The boundaries of NDC space are 0,1,0,1 for the left, right, bottom, and top boundaries respectively.
4. The values assigned are equal to the values last established with a SET DEVICE WINDOW statement. If no previous SET DEVICE WINDOW statement has been executed, the values assigned are the default boundaries of the device window.

xample

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE left_1,right_1,bottom,top
OPEN "RT31" FOR GRAPHICS AS DEVICE #1
SET WINDOW 0,100,0,100
SET TEXT HEIGHT 4

ASK DEVICE WINDOW #1 : left_1, right_1, bottom, top
GRAPH TEXT AT 10,80 : "Left boundary = "  + STR$(left_1)
GRAPH TEXT AT 10,70 : "Right boundary = " + STR$(right_1)
GRAPH TEXT AT 10,60 : "Bottom boundary = " + STR$(bottom)
GRAPH TEXT AT 10,50 : "Top boundary = " + STR$(top)
END
```

Output



```
Left boundary = 0

Right boundary = 1

Bottom boundary = 0

Top boundary = 1
```

ZK-4969-86

ASK LINE COLOR

The ASK LINE COLOR statement allows you to retrieve the current color index for lines.

Format

ASK LINE COLOR *int-var*

Syntax Rules

None.

Remarks

1. The range of valid color indices is device dependent. Results are unpredictable if you use the value assigned to *int-var* in a subsequent SET LINE COLOR statement and the index is undefined for any activated device.
2. On VT125 and VT240 terminals with the color option, the default value assigned to *int-var* represents one of the following colors:

Index	Color
0	Black
1	Green
2	Red
3	Blue

3. See also the SET LINE COLOR, SET/ASK COLOR MIX, and ASK MAX COLOR statements.

:xample

```
OPTION TYPE = EXPLICIT
DECLARE LONG color_var
ASK LINE COLOR color_var
SET LINE COLOR color_var + 2
SET LINE SIZE 5
GRAPH LINES 0.0,0.5; 1.0,0.5
SET LINE COLOR color_var
GRAPH LINES 0.0,0.2; 1.0,0.2
END
```

ASK LINE STYLE

The ASK LINE STYLE statement allows you to determine the current styl of lines.

Format

ASK LINE STYLE *int-var*

Syntax Rules

None.

Remarks

1. The numeric values retrieved for the line style represent the followin styles:

Value	Line Style
1	Solid (default)
2	Dashed
3	Dotted
4	Dashed-dotted

   Line styles greater than 4 are device dependent.
2. The default line style is 1, solid.
3. Line styles are device dependent. If you use the value assigned to *int-var* in a subsequent SET LINE STYLE statement and this value is not supported by any activated device, results are unpredictable.
4. For illustrations of the various line styles, see the SET LINE STYLE statement.

:xample

```
DECLARE LONG whats_my_line
!+
!Store the current line style
!-
ASK LINE STYLE whats_my_line
SET LINE STYLE 4
GRAPH LINES 0.4,0.8; 0.6,0.8
SET LINE STYLE 3
GRAPH LINES 0.4,0.7; 0.6,0.7
SET LINE STYLE 2
GRAPH LINES 0.4,0.6; 0.6,0.6
!+
!Regain the stored style value
!-
SET LINE STYLE whats_my_line
GRAPH LINES 0.5,0.5; 0.5,0.0
END
```

ASK MAX COLOR

The ASK MAX COLOR statement retrieves the maximum color index value for the specified device.

Format

ASK MAX COLOR *[#dev-id :] num-var*

Syntax Rules

When specified, *dev-id* must be preceded by the number sign (#) and followed by a colon (:).

Remarks

1. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. *Num-var* retrieves the maximum color index. Colors from 0 to *num-var* are defined for the device. This statement is valid only for devices with contiguous color indices.
3. The range of color indices is device dependent. If you use the value assigned to *num-var* in a subsequent SET . . . COLOR statement and the index value is undefined for any activated device, results are unpredictable.

:xample

```
OPTION TYPE = EXPLICIT
DECLARE LONG highest_color
ASK MAX COLOR highest_color
SET TEXT COLOR highest_color
GRAPH TEXT AT 0,0.8 :                          &
      "This device supports colors from 0 to " + STR$(highest_color)
END
```

ASK MAX LINE SIZE

The ASK MAX LINE SIZE statement retrieves the maximum scale factor for the width of lines.

Format

ASK MAX LINE SIZE *[#dev-id :] num-var*

Syntax Rules

When specified, *dev-id* must be preceded by the number sign (#) and followed by a colon (:).

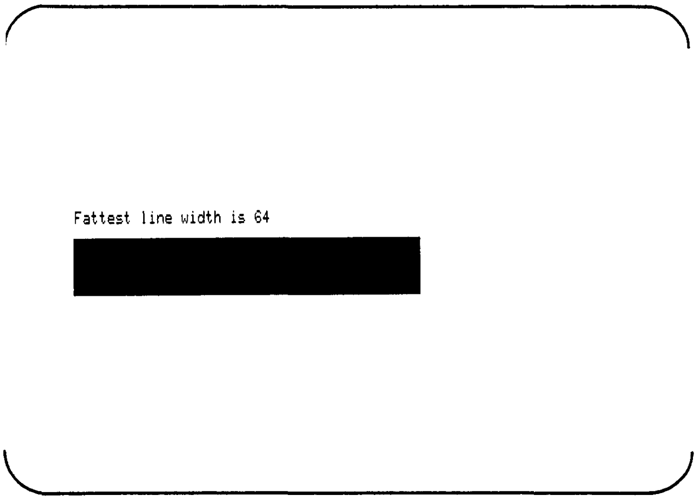
Remarks

1. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. *Num-var* retrieves the maximum line width scale factor.
3. The number of line sizes between 1 and *num-var* varies with each device.
4. The default value for the line width scale factor is 1.
5. You can use the value retrieved in *num-var* to set the line size with a SET LINE SIZE statement. However, if this value is undefined for a particular device, results are unpredictable.

:xample

```
OPTION TYPE = EXPLICIT
DECLARE LONG fattest
OPEN "TT55" FOR GRAPHICS AS DEVICE #1
ASK MAX LINE SIZE #1 : fattest
SET TEXT FONT -3, "STRING"
SET TEXT HEIGHT 0.03
GRAPH TEXT AT 0.1,0.5 :                                  &
      "Fattest line width is " + STR$(fattest)
SET LINE SIZE fattest
GRAPH LINES 0.1,0.4; 0.9,0.4
END
```

Output


```
Fattest line width is 64
```

ZK-4956-86

ASK MAX POINT SIZE

The ASK MAX POINT SIZE statement retrieves the maximum scale factor for POINT output.

Format

ASK MAX POINT SIZE *[#dev-id :] num-var*

Syntax Rules

1. When specified, *dev-id* must be preceded by a number sign (#) and followed by a colon (:).
2. *Num-var* retrieves the largest possible scale factor for point sizes.

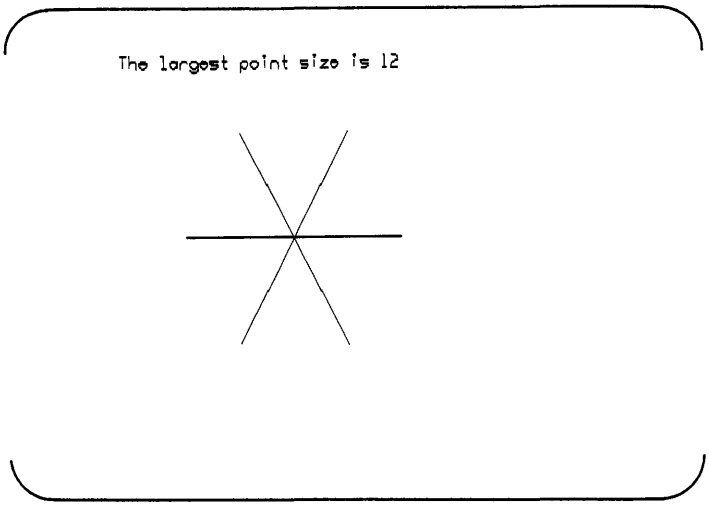
Remarks

1. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. Scale factors from 1 to *num-var* are available on the device specified. The number of different point sizes within the range is device dependent.
3. Points in the dot marker style are always drawn in the smallest possible size.
4. You can use the value retrieved in *num-var* to set the point size with a SET POINT SIZE statement. However, if the value is undefined for any activated device, results are unpredictable.

xample

```
DECLARE LONG largest
ASK MAX POINT SIZE largest
SET POINT SIZE largest
GRAPH TEXT AT 0.1,0.9 : "The largest point size is "+ STR$(largest)
GRAPH POINTS 0.5,0.5
```

Output



ZK-5517-86

ASK POINT COLOR

The ASK POINT COLOR statement allows you to retrieve the current color index used to display points.

Format

ASK POINT COLOR *int-var*

Syntax Rules

None.

Remarks

1. The range of valid color indices is device dependent. Results are unpredictable if you use the value assigned to *int-var* in a subsequent SET POINT COLOR statement and the index is undefined for any activated device.
2. On VT125 and VT240 terminals with the color option, the default value assigned to *int-var* represents one of the following colors:

Index	Color
0	Black
1	Green
2	Red
3	Blue

3. See also the SET POINT COLOR, SET/ASK COLOR MIX, and ASK MAX COLOR statements.

xample

```
OPTION TYPE = EXPLICIT
DECLARE LONG color_var
ASK POINT COLOR color_var
GRAPH POINTS 0.2,0.6
SET POINT COLOR (color_var + 1)
GRAPH POINTS 0.4,0.6
END
```

ASK POINT STYLE

The ASK POINT STYLE statement allows you to determine the current style of points.

Format

ASK POINT STYLE *int-var*

Syntax Rules

None.

Remarks

1. The numeric values retrieved for POINT represent the following styles:

Value	Point Style
1	Dot
2	Plus sign
3	Asterisk (default)
4	Circle
5	Diagonal cross

   These five styles are illustrated with the SET POINT STYLE statemen Values greater than 5 are device dependent.

2. The default point style is 3, an asterisk.
3. Point styles are device dependent. If you use the value assigned to *int-var* in a subsequent SET POINT STYLE statement and this value : not supported by any activated device, results are unpredictable.

xample

```
EXTERNAL PICTURE big_dipper
DECLARE LONG whats_the_point
!+
!Store the current point style
!-
ASK POINT STYLE whats_the_point
SET POINT STYLE 4
DRAW big_dipper
!+
!Regain the stored style value
!-
SET POINT STYLE whats_the_point
DRAW big_dipper
END
```

ASK STRING ECHO AREA

The ASK STRING ECHO AREA statement allows you to retrieve the boundaries of the current STRING echo area. The boundaries are assigne in device coordinates.

Format

$$\textbf{ASK STRING ECHO AREA} \left[\begin{array}{l} \#\textit{dev-id} \\ \textbf{, UNIT}\ \textit{int-exp} \end{array} \left\{ \; : \; \right\} \right]$$

real-var1, real-var2, real-var3, real-var4

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. *Real-var1* must be preceded by a colon if one or more optional clause are included.
3. *Real-var1, real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

Remarks

1. The echo area is the portion of your screen where the prompt appear: and where input can be supplied by a user. The default boundaries are device dependent. VAX BASIC assigns the boundaries for this are in device coordinates.

2. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
3. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the selection from a menu can be indicated with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
4. See also the SET STRING ECHO AREA statement.

xample

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE xmin,xmax,ymin,ymax,new_ymax
DECLARE STRING what
ASK STRING ECHO AREA xmin, xmax, ymin, ymax
!+
!Select larger echo boundary
!-
new_ymax = ymax * 2
SET STRING ECHO AREA xmin, xmax, ymin, new_ymax
LOCATE STRING what
    .
    .
    .
```

ASK TEXT ANGLE

The ASK TEXT ANGLE statement allows you to retrieve the current angl of text rotation.

Format

ASK TEXT ANGLE *real-var*

Syntax Rules

None.

Remarks

1. The default text angle is zero. The angle assigned to *real-var* is in radians or degrees; you can select the unit of measure with the OPTION ANGLE statement.
2. A positive nonzero value assigned to *real-var* indicates that subsequen displays of text are rotated in a counterclockwise direction; negative values indicate that subsequent displays of text are rotated in a clockwise direction.
3. Text is rotated about the starting point indicated in the GRAPH TEXT statement.
4. Angles that are integer multiples of 180° display text horizontally.
5. Angles that are integer multiples of 90° display text vertically.
6. For illustrations of various text angle settings, see the SET TEXT ANGLE statement.

xample

```
OPTION TYPE = EXPLICIT
OPTION ANGLE = DEGREES
DECLARE SINGLE turn
ASK TEXT ANGLE turn
SET TEXT ANGLE (turn + 45)
GRAPH TEXT AT 0.2,0.5 : "Where are we now?"
SET TEXT ANGLE turn
GRAPH TEXT AT 0.2,0.4 : "Back to where we started."
END
```

ASK TEXT COLOR

The ASK TEXT COLOR statements allow you to retrieve the current colo index used to draw text characters.

Format

ASK TEXT COLOR *int-var*

Syntax Rules

None.

Remarks

1. The range of valid color indices is device dependent. Results are unpredictable if you use the value assigned to *int-var* in a subsequen SET TEXT COLOR statement and the index is undefined for any activated device.
2. On VT125 and VT240 terminals with the color option, the default value assigned to *int-var* represents one of the following colors:

Index	Color
0	Black
1	Green
2	Red
3	Blue

3. See also the SET TEXT COLOR, SET/ASK COLOR MIX, and ASK MAX COLOR statements.

xample

```
OPTION TYPE = EXPLICIT
DECLARE LONG color_var
ASK TEXT COLOR color_var
GRAPH TEXT AT 0.0,0.6 : "This shows the default color."
SET TEXT COLOR color_var + 1
GRAPH TEXT AT 0.0,0.4 : "This is color index 2."
END
```

ASK TEXT EXPAND

The ASK TEXT EXPAND statement allows you to retrieve the current rati
of character width to height.

Format

ASK TEXT EXPAND *real-var*

Syntax Rules

None.

Remarks

1. The value retrieved is the expansion factor used for current GRAPH TEXT statements. This value can be set with the SET TEXT EXPAND statement.
2. The default height-to-width ratio is 1.0, which specifies the expansion factor defined in the font design.
3. For illustrations of various expansion settings, see the SET TEXT EXPAND statement.

xample

```
DECLARE SINGLE how_fat
!+
!Store current setting
!-
ASK TEXT EXPAND how_fat
SET TEXT EXPAND 2
GRAPH TEXT AT 0.1,0.9 : "This might be hard to read"
!+
!Use stored setting
!-
SET TEXT EXPAND how_fat
END
```

ASK TEXT EXTENT

The ASK TEXT EXTENT statement allows you to retrieve values for the boundaries of the text extent box for a given text string.

Format

ASK TEXT EXTENT *[#dev-id] , str-exp* **AT** *x-coord, y-coord* : *x-array, y-array*

Syntax Rules

1. The optional *dev-id* must be preceded by the number sign (#).
2. *Str-exp* contains the characters to be used for the text extent box.
3. *X-coord* and *y-coord* contain the world coordinates of the starting position for the text string.
4. *X-array* and *y-array* must each be a one-dimensional array containing at least four integer or real elements. Virtual arrays are not valid.

Remarks

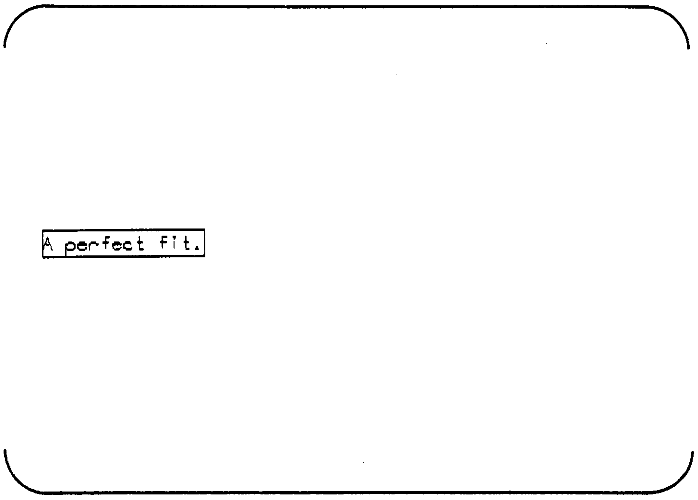
1. The optional *dev-id* identifies the device for which you want informa tion. If no identification clause is included, VAX BASIC uses a defaul identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASI(as the default device.
2. The text extent box is an imaginary rectangle that exactly surrounds the string expression when both the horizontal and vertical justification aspects are normal.
3. The boundaries of the text extent box are retrieved in world coordinates in *x-array* and *y-array*. This statement only retrieves the boundaries of the text box. To draw the text string, you must use a GRAPH TEXT statement, as shown in the example.

4. Current values for the text height, path, angle, and other text attributes are taken into consideration when VAX BASIC calculates the text extent box.
5. See also the ASK TEXT POINT statement.

Example

```
OPTION TYPE = EXPLICIT
DIM SINGLE x_coords(3), y_coords(3)
OPEN "my_term" FOR GRAPHICS AS DEVICE #1
SET WINDOW 0,100,0,100
SET TEXT HEIGHT 4
ASK TEXT EXTENT #1                          &
              , "A perfect fit." AT 5,50  &
              : x_coords, y_coords
GRAPH TEXT AT 5,50 : "A perfect fit."
!+
!Set area style to hollow
!and draw the text extent box
!-
SET AREA STYLE "HOLLOW"
MAT GRAPH AREA x_coords,y_coords
END
```

Output



ZK-5234-86

ASK TEXT FONT

The ASK TEXT FONT statement retrieves the number of the font and the value of the text precision.

Format

ASK TEXT FONT *int-var [, str-var]*

Syntax Rules

1. The value assigned to *int-var* specifies the font number.
2. The optional *str-var* specifies the level of precision with which the characters are drawn.

Remarks

1. The default font is number -1, which is displayed with stroke precision.
2. Hardware fonts are device dependent. Most devices support at least one hardware font; some support as many as six. When no hardware fonts are available, software fonts are used. Software fonts are device independent and provide a graphical representation of the defined characters.
3. Possible values for the level of precision are: STRING, CHAR, or STROKE. STRING provides the lowest precision, while STROKE provides the greatest. The precision value affects the representation of the text height and how text is clipped. For details and more examples, see the SET TEXT FONT, SET TEXT HEIGHT, and SET CLIP statements. Hardware fonts can be drawn with either STRING or CHAR precision; software fonts can only be drawn with STROKE precision.

ASK TEXT FONT

Example

```
DECLARE LONG this_font,                    &
        STRING precision
ASK TEXT FONT this_font, precision
GRAPH TEXT AT 0.1,0.8 : "This is font number " + STR$(this_font)
GRAPH TEXT AT 0.1,0.7 : "Precision is " + precision
END
```

ASK TEXT HEIGHT

The ASK TEXT HEIGHT statement retrieves the current value of the text height in world coordinates.

Format

ASK TEXT HEIGHT *real-var*

Syntax Rules

None.

Remarks

1. Height refers to the height of the uppercase letters in world coordinates.
2. The initial text height is 0.035. Adjustments to the world window usually require adjustments to the text height.
3. For more information, see the SET TEXT HEIGHT statement.

ASK TEXT HEIGHT

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE current_height
!+
!Ask for height in default scale
!-
ASK TEXT HEIGHT current_height
GRAPH TEXT AT 0.0,0.9 : "Current text height is " + STR$(current_height)
!+
!Change the world coordinate scale
!-
SET WINDOW 0,100,0,100
!+
!Set new height scale
!-
SET TEXT HEIGHT 5
GRAPH TEXT AT 0,60 :"Height is now 5/100  of"
GRAPH TEXT AT 0,40 :"the world window"
SET TEXT HEIGHT current_height
GRAPH TEXT AT 0,20 : "You won't be able to read this"
END
```

ASK TEXT JUSTIFY

The ASK TEXT JUSTIFY statement retrieves the current values of the horizontal and vertical components of text justification.

Format

ASK TEXT JUSTIFY *str-var1, str-var2*

Syntax Rules

1. The value for the horizontal component is assigned to *str-var1*.
2. The value for the vertical component is assigned to *str-var2*.

Remarks

1. At the start of program execution, the value for both the horizontal and vertical components is "NORMAL"; these are the default values until a SET TEXT JUSTIFY statement is executed.
2. Possible values for the horizontal component in *str-var1* are the following:

Value	Effect on Horizontal Component
LEFT	Corresponds to the left side of the text box passing through the text position
CENTER	Corresponds to the text position lying midway between the left and right sides of the text box
RIGHT	Corresponds to the right side of the text box passing through the text position
NORMAL	Depends on the text path—see below

3. Possible values for the vertical component in *str-var2* are the following:

Value	Effect on Vertical Component
TOP	The top of the text box passes through the text position.
CAP	The text position passes through the capline of the whole string.
HALF	The text position passes through the half-line of the whole string.
BASE	The text position lies on the baseline of the whole string.
BOTTOM	The bottom of the text box passes through the text.
NORMAL	See below.

4. NORMAL can be assigned for both the horizontal and vertical components. NORMAL provides a natural justification for each text path. For each of the text paths the horizontal and vertical components are as follows:

Text Path	Normal Horizontal	Normal Vertical
RIGHT	LEFT	BASE
LEFT	RIGHT	BASE
UP	CENTER	BASE
DOWN	CENTER	TOP

5. For more information and illustrations of various settings of text justification, see the SET TEXT JUSTIFY statement.

Example

```
DECLARE STRING horiz_justify, vert_justify
ASK TEXT JUSTIFY horiz_justify, vert_justify
IF horiz_justify = "NORMAL" AND vert_justify = "NORMAL"
   THEN
   SET TEXT JUSTIFY "LEFT" , "BOTTOM"
END IF
```

ASK TEXT PATH

The ASK TEXT PATH statement retrieves the current value for the direction of the text path.

ormat

ASK TEXT PATH *str-var*

yntax Rules

None.

emarks

1. Path refers to the direction in which the text is written relative to the text angle.
2. The value assigned to *str-var* is one of the following values:
   - RIGHT
   - LEFT
   - UP
   - DOWN
3. The default value for *str-exp* is RIGHT.
4. For illustrations of various text paths, see the SET TEXT PATH statement.

Example

```
OPTION TYPE = EXPLICIT
DECLARE STRING CONSTANT my_way = "LEFT"
DECLARE STRING which_way
   .
   .
   .
ASK TEXT PATH which_way
IF which_way = "RIGHT"
   THEN SET TEXT PATH my_way
END IF
   .
   .
   .
SET TEXT PATH which_way
   .
   .
   .
```

ISK TEXT POINT

The ASK TEXT POINT statement allows you to retrieve the coordinates for the concatenation point of a text string.

ormat

ASK TEXT POINT *[#dev-id] , str-exp* **AT** *x-coord1, y-coord1* : *x-coord2, y-coord2*

yntax Rules

1. The optional *dev-id* must be preceded by the number sign (#).
2. *Str-exp* contains the characters to be used for the text extent box.
3. *X-coord1* and *y-coord1* contain the world coordinates of the starting position for the text string.
4. The world coordinates of the concatenation point for the supplied text string are assigned to *x-coord2* and *y-coord2*.

emarks

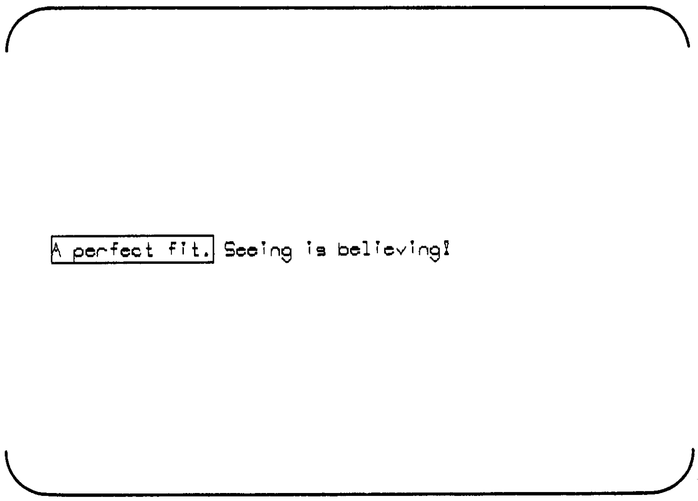
1. The concatenation point of a text string is the next appropriate point where a subsequent text string can be started.
2. This statement only retrieves the concatenation point. To draw the text string, you need to include a GRAPH TEXT statement.
3. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.

4. Current values for the text height, path, angle, and other text attributes are taken into consideration when VAX BASIC calculates the concatenation point.
5. See also the ASK TEXT EXTENT statement. For an additional example, see the SET TEXT COLOR statement.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE concat_x, concat_y
DIM SINGLE x_coords(3), y_coords(3)
OPEN "VT21" FOR GRAPHICS AS DEVICE #1
SET WINDOW 0,100,0,100
SET TEXT HEIGHT 4
ASK TEXT EXTENT #1 ,                    &
               "A perfect fit." AT 5,50 &
               : x_coords , y_coords
ASK TEXT POINT #1 ,                     &
               "A perfect fit." AT 5,50 &
               : concat_x, concat_y
GRAPH TEXT AT 5,50 : "A perfect fit."
!+
!Set area style to hollow
!-
SET AREA STYLE "HOLLOW"
!+
!Draw the text box
!-
MAT GRAPH AREA x_coords,y_coords
!+
!Add more text at the concatenation point
!-
GRAPH TEXT AT concat_x, concat_y : "Seeing is believing!"
END
```

Output


```
A perfect fit. Seeing is believing!
```
ZK-5237-86

ASK TEXT SPACE

The ASK TEXT SPACE statement retrieves the value for the current spacing between characters.

Format

ASK TEXT SPACE *real-var*

Syntax Rules

None.

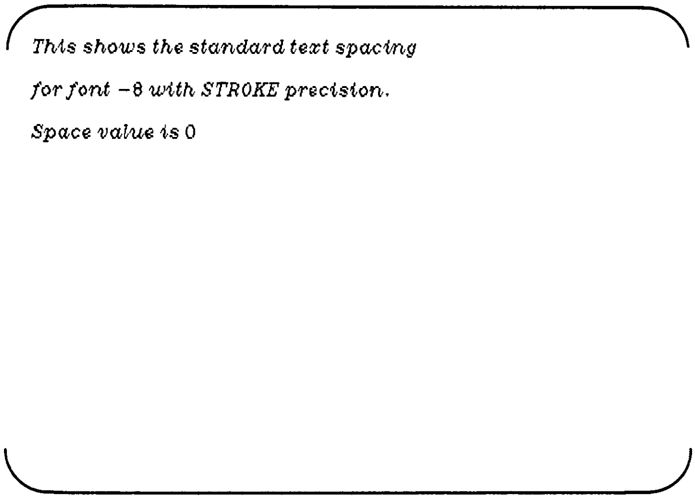
Remarks

1. Space refers to the distance between adjacent characters. The distanc is expressed as a fraction of the text height.
2. The default value for the spacing is 0, which produces adjacent characters spaced according to the font design.
3. A positive *real-var* value inserts more space between characters.
4. A negative *real-var* value causes characters to be closer together.
5. Subsequent GRAPH TEXT statements use the spacing value indicated by *real-var* until a SET TEXT SPACE statement is executed.

xample

```
DECLARE SINGLE spaced_out
ASK TEXT SPACE spaced_out
SET TEXT FONT -8, "STROKE"
SET TEXT HEIGHT 0.04
GRAPH TEXT AT 0.0,0.9 : "This shows the standard text spacing"
GRAPH TEXT AT 0.0,0.8 : "for font -8 with STROKE precision."
GRAPH TEXT AT 0.0,0.7 : "Space value is " + STR$(spaced_out)
```

Output



```
This shows the standard text spacing

for font -8 with STROKE precision.

Space value is 0
```

ZK-4880-86

ASK TRANSFORMATION

The ASK TRANSFORMATION statement lets you determine which transformation is currently being used to map world coordinate points onto the world viewport in NDC space.

Format

ASK TRANSFORMATION *int-var*

Syntax Rules

The value assigned to *int-var* is an integer between 1 and 255.

Remarks

1. The transformation assigned to *int-var* is the transformation currently used to transform world coordinate points for display onto the world viewport in NDC space.
2. All subsequent graphics output and input use this transformation number until a different number is specified in a SET TRANSFORMATION, SET VIEWPORT, or SET WINDOW statement. If none of these statements has been executed, the current transformation is 1.
3. See also the SET INPUT PRIORITY statement.

xample

```
DECLARE LONG which_tran
    .
    .
    .
ASK TRANSFORMATION which_tran
IF which_tran = 1
   THEN SET TRANSFORMATION 2
   ELSE GOTO Set_up_screens
END IF
    .
    .
    .
```

ASK TRANSFORMATION LIST

The ASK TRANSFORMATION LIST statement retrieves the list of define transformation numbers, starting with the transformation with the highes priority for input.

Format

ASK TRANSFORMATION LIST *[* **, COUNT** *int-var* : *] int-array*

Syntax Rules

1. When specified, the COUNT clause must be followed by a colon (:).
2. *Int-array* must be a one-dimensional array containing integer or real data types. Virtual arrays are invalid.

Remarks

1. The values assigned to *int-array* are transformation numbers for all defined transformations. The list is ordered by input priority, starting with the transformation with the highest priority.
2. *Int-array* can be larger than the actual number of transformations. If the number of defined transformations is larger than the size of *int-array,* the actual number of defined transformations can be retrieved i *int-var* with the optional COUNT clause.
3. For more information about input priorities, see the SET INPUT PRIORITY statement.

xample

```
DECLARE LONG how_many
DIM LONG xforms(15)
    .
    .
    .
ASK TRANSFORMATION LIST , COUNT how_many : xforms
    .
    .
    .
```

ASK VALUE ECHO AREA

The ASK VALUE ECHO AREA statement allows you to retrieve the boundaries of the current VALUE echo area. The boundaries are assigned in device coordinates.

Format

```
                       [ #dev-id          {   }  ]
ASK VALUE ECHO AREA    [ , UNIT int-exp   { : }  ]
                       [                  {   }  ]
                    real-var1, real-var2, real-var3, real-var4
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. *Real-var1* must be preceded by a colon if one or more optional clause are included.
3. *Real-var1, real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

Remarks

1. The echo area is the portion of your screen where the prompt appear and where input can be supplied by a user. The default boundaries are device dependent.
2. VAX BASIC assigns the boundaries for this area in device coordinates

3. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
4. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
5. See also the SET VALUE ECHO AREA statement.

:xample

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE xmin,xmax,ymin,ymax,                                 &
               new_xmin,new_xmax,new_ymin,new_ymax,what
ASK VALUE ECHO AREA xmin, xmax, ymin, ymax
!+
!Select echo boundary on bottom right of screen
!-
new_xmin = 0
new_xmax = xmin/2
new_ymin = ymin
new_ymax = ymax/2
SET VALUE ECHO AREA new_xmin, new_xmax, new_ymin, new_ymax
LOCATE VALUE what
    .
    .
    .
```

ASK VIEWPORT

The ASK VIEWPORT statement retrieves the current values for the boundaries of the world viewport.

Format

ASK VIEWPORT *[* **, TRAN** *int-exp* : *] real-var1, real-var2, real-var3, real-var4*

Syntax Rules

1. When included, the optional TRAN clause must be followed by a colon (:).
2. *Int-exp* must be between 1 and 255.
3. *Real-var1, real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

Remarks

1. If more than one transformation is defined, you can use the optional TRAN clause to specify the transformation you want information about. If the TRAN clause is not specified, the viewport boundaries for the default transformation are assigned.
2. If no SET VIEWPORT statement has been executed, the boundary values assigned are for the default world viewport.
3. The default world viewport is the entire NDC space. The boundaries of NDC space are 0,1,0,1 for the left, right, bottom, and top boundaries respectively.
4. See also the SET VIEWPORT statement.

xample

```
DECLARE LONG left_1,right_1,bottom,top
    .
    .
    .
ASK VIEWPORT , TRAN 2 : left_1,right_1,bottom,top
!+
!Use TRAN 2 boundaries to set a new viewport for TRAN 4
!-
SET VIEWPORT , TRAN 4 : left_1,right_1/2,bottom,top/2
    .
    .
    .
```

ASK WINDOW

The ASK WINDOW statement retrieves the current values for the boundaries of the world window.

Format

ASK WINDOW *[* **, TRAN** *int-exp* : *] real-var1, real-var2, real-var3, real-var4*

Syntax Rules

1. When included, the optional TRAN clause must be followed by a colon (:).
2. *Int-exp* must be between 1 and 255.
3. *Real-var1, real-var2, real-var3,* and *real-var4* must be floating-point variables to represent the left, right, bottom, and top boundaries respectively.

Remarks

1. If more than one transformation is defined, you can use the optional TRAN clause to specify the window you want information about. If the TRAN clause is not specified, the boundaries for the default window (TRAN 1) are retrieved.
2. If no SET WINDOW statement has been executed, the boundary values assigned are for the default world window.
3. The default world window boundaries are 0,1,0,1 for the left, right, bottom, and top boundaries respectively.
4. For more information about the world window, see the SET WINDOV statement and Chapter 5 in this manual.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE sine_curve,fudge_it
DECLARE LONG left_1,right_1,bottom,top
   .
   .
   .
ASK WINDOW, TRAN 2 : left_1, right_1, bottom, top
IF bottom <= -1 AND top >= 1
   THEN
       DRAW sine_curve
   ELSE
       DRAW fudge_it
END IF
END
```

CLEAR

The CLEAR statement clears graphics text and images from the screen.

Format

CLEAR *[#dev-id]*

Syntax Rules

The optional *dev-id* must be preceded by a number sign (#).

Remarks

1. The optional *dev-id* identifies the device for which you want information. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. Output devices are not implicitly cleared before graphics output statements are executed. If you do not clear a display, subsequent displays are overlaid on any previous output from the program.
3. Output devices are implicitly cleared when they are opened.

xample

```
EXTERNAL PICTURE Select_menu, sub_menu
OPEN "term_name" FOR GRAPHICS AS DEVICE #1
OPEN "VT231" FOR GRAPHICS AS DEVICE #2
OPEN "rec_term" FOR GRAPHICS AS DEVICE #3
!+
!Select_menu displayed on all activated devices
!-
DRAW Select_menu
!+
!Clear the display on device #1 only
!-
CLEAR #1
!+
!Select-menu display still active on devices #2 and #3
!Select-menu overlaid with sub-menu on devices #2 and #3
!Sub-menu displayed by itself on device #1
!-
DRAW sub_menu
    .
    .
    .
```

CLOSE DEVICE

The CLOSE DEVICE statement allows you to explicitly close a specified device. Subsequent graphics output will not be displayed on this device.

Format

CLOSE DEVICE *#dev-id*

Syntax Rules

You must supply the number sign (#) with *dev-id*.

Remarks

1. When a device is closed, graphics output cannot be displayed on that device. However, graphics output is displayed at all other active devices.
2. VAX BASIC implicitly closes all open devices when a graphics program terminates.
3. VAX BASIC implicitly deactivates the specified device if it is active when the CLOSE DEVICE statement is executed.
4. If the default device is closed and no other device is open, VAX BASIC reopens the default device when subsequent graphics output statements are executed.
5. If you intend to open the same device later during program execution, it is more efficient to deactivate the device rather than close it.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE US_political, US_relief
OPEN "RT47" FOR GRAPHICS AS DEVICE #1
OPEN "VTA247" FOR GRAPHICS AS DEVICE #2
OPEN "my_term" FOR GRAPHICS AS DEVICE #3
!+
!Display metafile on all devices
!-
GRAPH METAFILE "states_outline.pic"
!+
!Close device #1
!-
CLOSE DEVICE #1
!+
!Deactivate, rather than close, device #2
!-
DEACTIVATE DEVICE #2
!+
!Draw picture on device #3 only
!-
DRAW US_political
!+
!Reactivate device #2
ACTIVATE DEVICE #2
CLEAR #3
!+
!Display picture on devices #2 and #3
!-
DRAW US_relief
END
```

DEACTIVATE DEVICE

The DEACTIVATE DEVICE statement deactivates the specified output device. Subsequent graphics output will not be displayed on this device.

Format

DEACTIVATE DEVICE *#dev-id*

Syntax Rules

You must supply the number sign (#) with *dev-id*.

Remarks

1. This statement deactivates, but does not close, the specified device.
2. This statement is ignored if the specified output device is not activated.
3. If the default device (#0) is deactivated and subsequent graphics output statements are executed, VAX BASIC displays the output at all other active devices. If no other devices are activated, VAX BASIC reactivates the default device for output.
4. See also the ACTIVATE DEVICE statement.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE Sine_curve, ellipse
DIM SINGLE x_array(20),y_array(20),                 &
           x_curve(12),y_curve(12)
OPEN "BASIC_term" FOR GRAPHICS AS DEVICE #1
OPEN "my_vax" FOR GRAPHICS AS DEVICE #2 , TYPE 41
!+
!Display on all active devices
!-
DRAW ellipse
DEACTIVATE DEVICE #2
!+
!Display on all active devices except #2
!-
DRAW sine_curve
ACTIVATE DEVICE #2
!+
!Display on all devices
!-
GRAPH TEXT AT 0.5,0.5 :"Display on all open devices"
```

DRAW

The DRAW statement invokes the specified PICTURE subprogram. Optional transformation functions allow you to invoke the picture with altered world coordinates.

Format

DRAW *pic-name [(param-list)] [* **WITH** *tran-term [* tran-term]...]*

tran-term:

$$\left\{\begin{array}{l}\textbf{SCALE}\,(\textit{real-exp [, real-exp]})\\ \textbf{SHEAR}\,(\textit{angle})\\ \textbf{SHIFT}\,(\textit{real-exp, real-exp})\\ \textbf{ROTATE}\,(\textit{angle})\\ \textbf{TRANSFORM}\\ \textit{num-matrix}\end{array}\right\}$$

Syntax Rules

1. *Pic-name* must be a valid VAX BASIC picture name.
2. Parameters in the *param-list* must agree in number and data type with the parameters in a PICTURE statement.
3. *Tran-term* must be one of the VAX BASIC transformation functions listed or a valid user-supplied transformation matrix.
4. The value for *angle* supplied with the ROTATE and SHIFT functions can be in radians or degrees, as specified in the OPTION ANGLE statement.

5. When DRAW WITH *num-matrix* is used, *num-matrix* must be a two-dimensional numeric matrix with lower bounds of zero and upper bounds of 4 in both directions. VAX BASIC signals a compile-time error when the compiler detects a nonzero-based matrix; otherwise, a run-time error is signaled. *Num-matrix* cannot be a packed decimal or a virtual array.

emarks

1. Pictures displayed with the DRAW statement should be declared with the EXTERNAL statement.
2. DRAW statements can be included in a picture to invoke another picture or to invoke the picture recursively.
3. Graphics objects displayed within a picture with PLOT are affected by transformation functions included in the DRAW statement that invokes the picture; objects displayed with GRAPH are *not* affected by transformation functions.
4. When text is drawn in a picture with GRAPH TEXT , the text attributes are not automatically adjusted according to the transformation functions on the DRAW statement that invokes the picture. However, the text height and text starting point can be adjusted with the TRANSFORM function for SHIFT and positive SCALE transformations. For an example on how to do this, see the TRANSFORM function in this chapter.
5. The order of the transformation functions can affect the results of transformations when more than one function is included because transformation functions are not associative. VAX BASIC transforms points in the order of the transformation functions in the DRAW statement; therefore SHIFT(2,1) * SCALE(3) does not result in the same coordinate values as SCALE(3) * SHIFT(2,1).
6. **SCALE**

   The SCALE function multiplies the coordinates of point x,y by the arguments supplied. SCALE(A,B) changes the coordinates of the point to A*x,B*y. SCALE(A) changes the coordinates of a point to A*x,A*y. Arguments can be less than 1 to effect a decrease in size.

The resulting matrix for SCALE(A,B) is as follows:

```
A 0 0 0
0 B 0 0
0 0 1 0
0 0 0 1
```

When you supply just one argument with the SCALE function, both the x- and y-coordinates of a point are multiplied by the same argument. To affect a change in only one dimension, you can supply 1 as the second argument, for example SCALE(0.5,1), or SCALE(1,2).

See Example 1.

7. **SHEAR**

The SHEAR function changes the coordinates of point x,y so that the resulting point leans forward by the angle specified. The resulting point is (x+y * TAN(A)),y. Arguments can be negative.

The SHEAR function requires an argument in degrees or radians; yo can select the unit of measure with the OPTION ANGLE statement.

The resulting matrix for SHEAR(A) is as follows:

```
     1  0 0 0
TAN(A)  1 0 0
     0  0 1 0
     0  0 0 1
```

See Example 2.

8. **SHIFT**

The SHIFT function moves the point x,y to the resulting point x+A,y+B. Arguments can be negative.

The resulting matrix for SHIFT(A,B) is as follows:

```
1 0 0 0
0 1 0 0
0 0 1 0
A B 0 1
```

See Example 3.

9. **ROTATE**

Like SHEAR, the ROTATE function requires an argument in degrees or radians. You select the unit of measure with the OPTION ANGL statement; the default is radians. A positive argument rotates the point x,y in a counterclockwise direction about the point of origin of the world coordinates. A negative value rotates the point in a clockwise direction. You can alter the point of origin by setting the window such that the point 0,0 is at the center of the window.

The resulting matrix for ROTATE(A) is as follows:

```
 COS(A) SIN(A) 0 0
-SIN(A) COS(A) 0 0
   0      0    1 0
   0      0    0 1
```

See Example 4.

10. **TRANSFORM**

    The TRANSFORM function returns the cumulative transformations for all transformations in current picture invocations. TRANSFORM takes no arguments. Note that current transformations are passed to pictures by default; therefore, when you invoke a picture with the TRANSFORM function on the DRAW statement, the transformations in the DRAW statement are applied twice.

    See Example 5.

11. **Num-matrix**

    You can specify your own two-dimensional matrix to substitute for a predefined VAX BASIC transformation function. The matrix must have lower bounds of zero and upper bounds of 4 in both directions. Row and column zero should not be assigned values as VAX BASIC uses these elements during matrix manipulation. For instance, rather than use a combined transformation such as SHIFT * SHEAR, you can assign the combination matrix to your own array:

```
MAT my_way = SHIFT(2,1) * SHEAR(25)
   .
   .
   .
DRAW circle WITH my_way
```

    Similarly, you can initialize an array with your own choice of values and use this with the DRAW statement:

```
DRAW circle WITH evs_way
```

    Note that when a transformation function consists of a numeric array that is not a 4 by 4 matrix, a fatal error is signaled. VAX BASIC signals a compile-time error when the compiler detects a nonzero based matrix; otherwise, a run-time error is signaled.

A fatal error is signaled when VAX BASIC cannot compute the invers of the *num-matrix* in a DRAW statement and points in a picture are input using the GET or MAT GET statement. See Chapter 7 for information about input within picture definitions.

See Example 6.

12. Transformation functions in DRAW statements within a picture are cumulative. That is, when a second or subsequent picture is invoked within a picture definition, the transformations from all of the relevan DRAW statements are applied to the points generated by the second c subsequent picture. For example, in the following picture, the picture *nest2* is invoked with a DRAW statement equivalent to DRAW WITH SHIFT(50,0) * SCALE(2,1).

Example

```
PICTURE nest 1
EXTERNAL PICTURE nest2
   .
   .
   .
   DRAW nest2 WITH SCALE(2,1)
END PICTURE
!+
!Invoke picture nest1
!-
DRAW nest1 WITH SHIFT(50,0)
```

See the TRANSFORM function for more information on accumulated transformations.

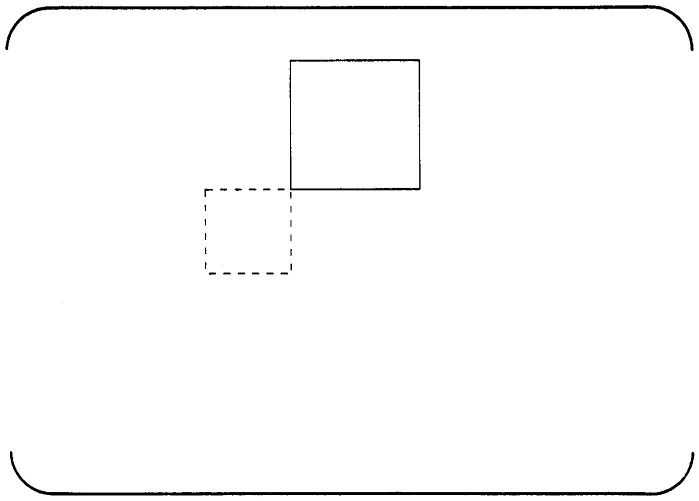
Examples

Example 1

```
EXTERNAL PICTURE box(LONG)
DRAW box(3)
DRAW box(1) WITH SCALE(1.5,1.5)
END
```

```
PICTURE box(LONG line_style)
SET LINE STYLE line_style
PLOT LINES 0.4,0.6;                 &
           0.6,0.6;                 &
           0.6,0.4;                 &
           0.4,0.4;                 &
           0.4,0.6
END PICTURE
```

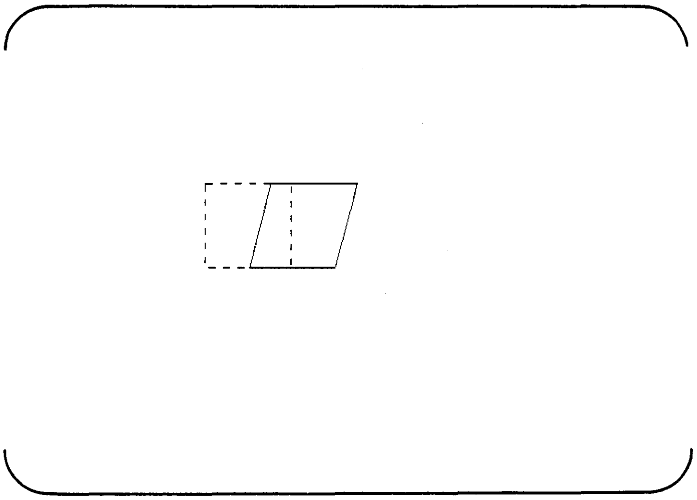
Output 1



Example 2

```
EXTERNAL PICTURE box(LONG)
DRAW box(3)
DRAW box(1) WITH SHEAR(0.25)
END
```

Output 2

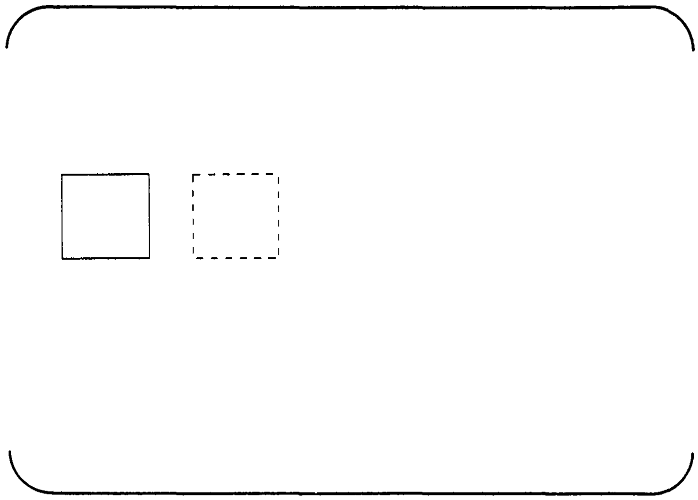


ZK-4957-86

Example 3

```
EXTERNAL PICTURE box(LONG)
DRAW box(3)
DRAW box(1) WITH SHIFT(-0.3,0)
END
```

Output 3

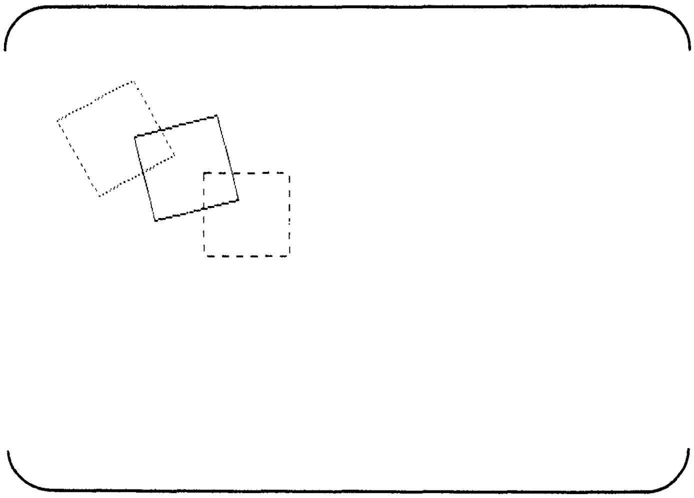


ZK-4963-86

Example 4

```
OPTION ANGLE = RADIANS
EXTERNAL PICTURE box(LONG)
DRAW box(3)
DRAW box(4) WITH ROTATE(0.25)
DRAW box(1) WITH ROTATE(0.5)
END
```

Output 4



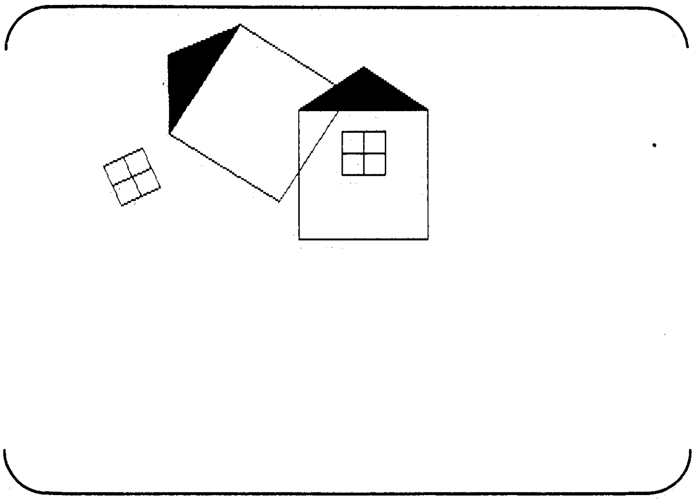
ZK-4983-86

Example 5

```
PROGRAM demo_transform
  OPTION ANGLE = RADIANS
  EXTERNAL PICTURE house
  SET WINDOW -1,1,-1,1
  DRAW house
  DRAW house WITH ROTATE(1)
END PROGRAM

PICTURE house
  EXTERNAL PICTURE look_out
  PLOT LINES : 0.2,0.0; 0.8,0.0;              &
               0.8,0.6; 0.2,0.6;              &
               0.2,0.0
  PLOT AREA 0.2,0.6; 0.5,0.8; 0.8,0.6
  DRAW look_out WITH TRANSFORM
END PICTURE
```

```
PICTURE look_out
  PLOT LINES : 0.4,0.3; 0.4,0.5; 0.6,0.5;     &
             0.6,0.3; 0.4,0.3
  PLOT LINES 0.4,0.4; 0.6,0.4
  PLOT LINES 0.5,0.3; 0.5,0.5
END PICTURE
```

Output 5

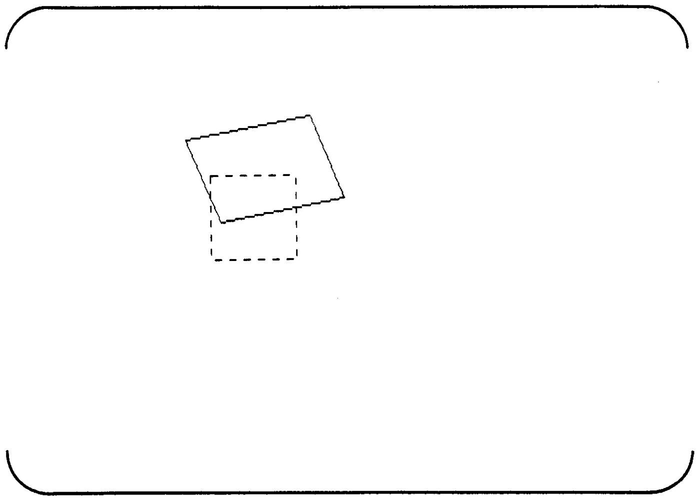


ZK-4974-86

Example 6

```
OPTION ANGLE = DEGREES
EXTERNAL PICTURE box(LONG)
DECLARE SINGLE prog_matrix(4,4)
MAT prog_matrix = ROTATE(15) * SCALE(1.5,1)
DRAW box(3)
DRAW box(1) WITH prog_matrix
END
```

Output 6


ZK-4966-86

ND PICTURE

The END PICTURE statement marks the end of a PICTURE subprogram.

ormat

END PICTURE

yntax Rules

None.

emarks

1. When an END PICTURE statement is executed, execution of the picture is terminated. Control is returned to the statement following the DRAW statement that invoked the picture.
2. See also the EXIT PICTURE statement.

xample

```
PICTURE sample
  EXTERNAL PICTURE box(LONG)
  DRAW box(3)
  GRAPH TEXT AT 0.1,0.8 : "One Way Out"
END PICTURE
```

EXIT PICTURE

The EXIT PICTURE statement allows you to terminate execution of a PICTURE subprogram.

Format

EXIT PICTURE

Syntax Rules

None.

Remarks

1. When an EXIT PICTURE statement is executed, execution of the picture is terminated. Control is returned to the statement following the DRAW statement that invoked the picture.
2. See also the END PICTURE statement.

Example

```
PROGRAM draw_sample
  EXTERNAL PICTURE sample(LONG)
  DRAW sample(100)
END PROGRAM

PICTURE sample(LONG pass)
  IF pass = 100%
     THEN
     GRAPH TEXT AT 0.1,0.7 : "Emergency exit"
     EXIT PICTURE
  END IF
  GRAPH TEXT AT 0.1,0.6 : "Exit ahead"
END PICTURE
```

GET POINT

A GET POINT statement accepts a single point of user input.

Format

```
            ┌ #dev-id                                       ┌   ┐ ┐
GET POINT   │ , UNIT int-exp                                │ : │ │
            └ , AT x-coord, y-coord [ USING TRAN int-exp]   └   ┘ ┘
          x-coord, y-coord [ , int-var]
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. *Int-exp* must be an integer between 1 and 255.
3. If one or more optional clauses are included, one colon (:) is required before *x-coord*.

Remarks

1. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. The user-supplied input is interpreted using the transformation with the highest priority that contains the point within its viewport. The world coordinates of the point are assigned to *x-coord* and *y-coord*. VAX BASIC assigns the number of the transformation actually used to the optional *int-var*.

3. An initial point can be specified with the optional AT clause. If an initial point is specified in the GET POINT statement, this point overrides an initial point previously set with the SET INITIAL POINT statement.
4. The world coordinates you supply for the initial point are interpreted with the transformation that is currently selected for output. You can specify an alternative transformation for interpreting this point in the optional USING TRAN clause.
5. The transformation with the highest priority is 1, unless a SET INPUT PRIORITY, SET WINDOW, SET VIEWPORT, or SET TRANSFORMATION statement has been executed.
6. Points accepted with the GET POINT statement in a PICTURE subprogram are transformed through the *inverse* of any transformation functions on the DRAW statement that invoked the picture. For example, if a picture is invoked with SCALE(2), device coordinates of input points are transformed to world coordinates and then divided by two (the inverse of SCALE(2)).

   Points accepted by the LOCATE POINT statement, however, are not affected by the transformation functions. See the example for the MAT GET POINTS statement for an illustration of this difference.

   Note that a fatal error is signaled when VAX BASIC cannot compute the inverse of a user-supplied matrix in a DRAW statement and points in a picture are input using the GET or MAT GET statement.
7. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.

Example

```
DECLARE LONG which_tran                          &
       ,SINGLE user_x,user_y
OPEN "RT13" FOR GRAPHICS AS DEVICE #1
OPEN "VT101" FOR GRAPHICS AS DEVICE #3
SET WINDOW , TRAN 2 : 0,100,0,100
SET VIEWPORT , TRAN 2 : 0,0.5,0,0.5
!+
!Use TRAN 1 for initial point
!-
GET POINT #3 , AT 50,50 USING TRAN 1 : user_x, user_y ,which_tran
!+
!Use TRAN 2 (current) for initial point
!-
GET POINT #1 , AT 0.5,0.5 : user_x, user_y , which_tran
IF which_tran <> 2
   THEN GOTO User_help
END IF
   .
   .
   .
```

GRAPH

The GRAPH statement draws the specified object on all active devices. The sequence is determined by the coordinates supplied in the point list. Values for the attributes such as color and style use the current settings.

Format

$$\text{GRAPH} \quad \left\{ \begin{array}{l} \text{POINTS} \\ \text{LINES} \\ \text{AREA} \end{array} \right\} \textit{[:] x-coord , y-coord [; x-coord, y-coord]...}$$

Syntax Rules

1. A GRAPH POINTS statement must include the coordinates for at leasi one point.
2. A GRAPH LINES statement requires the coordinates for a minimum of two points.
3. A GRAPH AREA statement requires the coordinates for a minimum oi three points.

Remarks

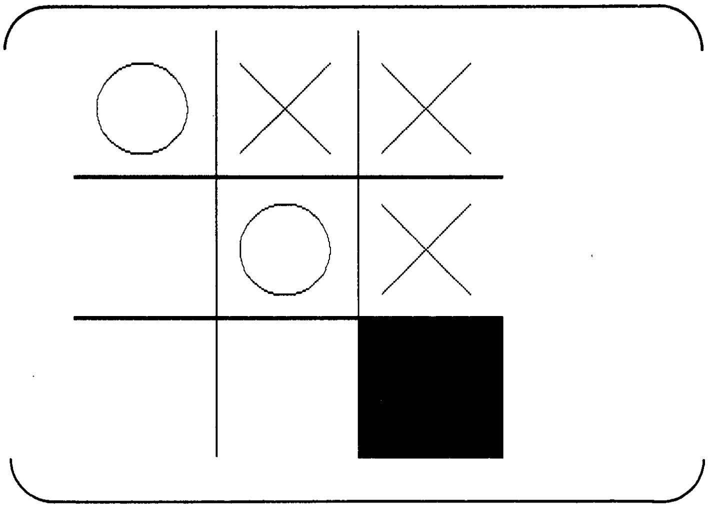
1. The sequence of the objects drawn is determined by the order of the coordinate pairs in the GRAPH statement; the first coordinate pair designates the first point, the second pair designates the second point, and so on through the end of the coordinate pairs.
2. If GRAPH LINES is used to draw a closed figure, the coordinates of the first point should be supplied again for the last point. For example, to draw a square with GRAPH LINES, you must supply the four points of the square and then supply the first point again as the fifth point.

3. To draw closed areas with the GRAPH AREA statement, you supply the same number of points as there are vertices in the polygon.
4. The coordinates of points in GRAPH statements within a picture definition are not affected by the transformations in the DRAW statement that invokes the picture. To transform points within a picture definition, you must use the PLOT statement. However, when you know that transformation functions will not be used, use GRAPH rather than PLOT statements, because they are more efficient.

Example

```
PROGRAM Tick_tack
   .
   .
   .
!+
!Set up game board
!-
SET LINE STYLE 1
SET LINE SIZE 2
SET LINE COLOR 3
GRAPH LINES 0,0.33; 1,0.33
GRAPH LINES 0,0.66; 1,0.66
GRAPH LINES 0.33,0; 0.33,1
GRAPH LINES 0.66,0; 0.66,1
!+
!Place X markers
!-
SET POINT COLOR 1
SET POINT STYLE 5
SET POINT SIZE 5
GRAPH POINTS 0.49,0.82; 0.82,0.82; 0.82,0.49
!+
!Place circle markers
!-
SET POINT COLOR 3
SET POINT STYLE 4
GRAPH POINTS 0.16,0.82; 0.49,0.49
!+
!Mark next position
!-
SET AREA COLOR 1
SET AREA STYLE "SOLID"
GRAPH AREA 0.66,0; 1,0; 1,0.33; 0.66,0.33
END PROGRAM
```

Output



ZK-5526-86

GRAPH METAFILE

The GRAPH METAFILE statement generates graphics output as specified in the metafile records.

Format

GRAPH METAFILE *file-spec*

Syntax Rules

1. *File-spec* must be a string expression containing a valid metafile.
2. If a metafile is opened with an OPEN . . . FOR GRAPHICS statement, the *file-spec* in the OPEN . . . FOR GRAPHICS statement must match the file specification in the GRAPH METAFILE statement.

Remarks

1. After a metafile has been created, it can be displayed with the GRAPH METAFILE statement.
2. You can substitute a logical name for the full metafile specification.
3. A GRAPH METAFILE statement implicitly opens and activates the metafile before generating the output.
4. To create a metafile, you send graphics output from your program to a file. You must open the file as device type 2 with the OPEN . . . FOR GRAPHICS statement.
5. GRAPH METAFILE displays the output on all activated devices. It is not necessary to open a metafile when you display it.

Examples

Example 1

```
PROGRAM create_file
!+
!Open and send output to an output metafile
!-
OPEN "[MCKAY]swan.pic" FOR GRAPHICS AS DEVICE #1 , TYPE 2
EXTERNAL PICTURE swan
DRAW swan
END PROGRAM
```

Example 2

```
PROGRAM draw_file
!+
!Display contents of metafile on default device
!-
GRAPH METAFILE "[MCKAY]swan.pic"
END PROGRAM
```

GRAPH TEXT

The GRAPH TEXT statement draws the specified characters. The display is governed by the current values of the text attributes HEIGHT, JUSTIFY, ANGLE, SPACE, EXPAND, FONT and PRECISION.

Format

GRAPH TEXT *[,]* **AT** *x-coord* **,** *y-coord* **:** *str-exp*

Syntax Rules

1. *X-coord* and *y-coord* specify the starting position of the text display.
2. *Str-exp* contains the characters to be drawn.

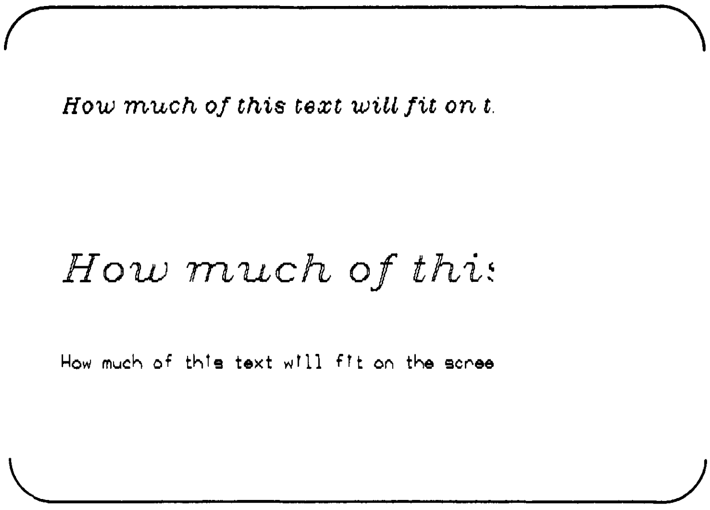
Remarks

1. The text string in *str-exp* is not wrapped on the terminal screen; text that extends beyond the device viewport is not visible. Text is also clipped at the world viewport by default. For more information, see the SET/ASK CLIP statements.
2. The maximum length of the text string is dependent on the setting of the various text attributes and the amount of display space available. See the example.
3. Text attributes are not automatically adjusted according to the transformation functions on a DRAW statement when text is drawn within a picture. However, the text height and text starting point can be adjusted with the TRANSFORM function. See the example for the TRANSFORM function in this chapter.
4. You can format graphics text with the FORMAT$ function.
5. After execution of a SET WINDOW statement, you may need to change the text height with the SET TEXT HEIGHT statement.

Example

```
DECLARE STRING CONSTANT text_string =                           &
                        "How much of this will fit on the screen?"
DECLARE SINGLE def_font, def_height,                            &
        STRING def_precision
!+
!Store default values
!-
ASK TEXT FONT def_font, def_precision
ASK TEXT HEIGHT def_height

SET TEXT FONT -8% , "STROKE"
SET TEXT HEIGHT 0.05
GRAPH TEXT AT 0.0,0.8 : text_string
!+
!Increase the text height
!-
SET TEXT HEIGHT 0.1
GRAPH TEXT AT 0.0,0.4 : text_string
!+
!Restore the default text attributes
!-
SET TEXT HEIGHT def_height
SET TEXT FONT def_font , def_precision
GRAPH TEXT AT 0.0,0.2 : text_string
END
```

Output



```
How much of this text will fit on t

How much of this

How much of this text will fit on the scree
```

ZK-5518-86

LOCATE CHOICE

The LOCATE CHOICE statement accepts a user's selection from a menu.

Format

$$\textbf{LOCATE CHOICE} \left[\begin{array}{l} \#\textit{dev-id} \\ \textbf{, UNIT } \textit{int-exp1} \\ \textbf{, INITIAL } \textit{int-exp2} \end{array} \left\{ : \right\} \right] \textit{int-var}$$

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by a number sign (#).
2. If one or more optional clauses are included, one colon (:) is required before *int-var*.

Remarks

1. Execution of a LOCATE CHOICE statement displays the menu to the user. The menu is displayed until the user inputs a menu selection. Program control continues with the statement following the LOCATE CHOICE statement.
2. The menu item input by the user is assigned to *int-var*.
3. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.

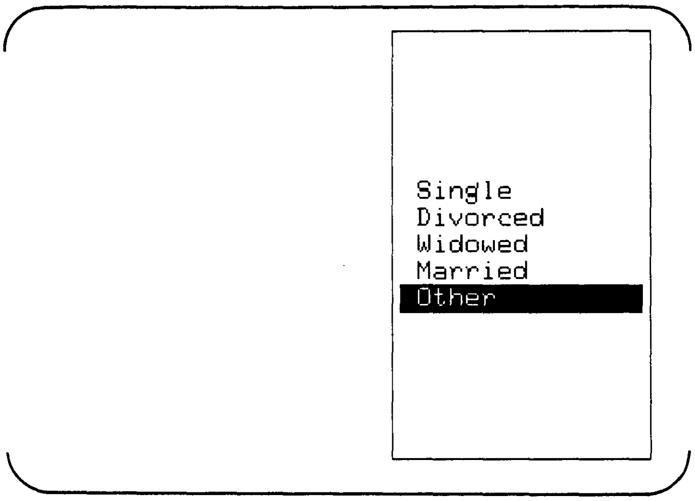
4. The optional INITIAL clause allows you to specify an initial response for the user with *int-exp2*. This initial value is highlighted in the echo area. The features of menu displays are device dependent.
5. The INITIAL clause in a LOCATE CHOICE statement overrides the INITIAL clause on a previous SET INITIAL CHOICE statement.
6. When no INITIAL clause is included, the initial choice established with the last SET INITIAL CHOICE statement is used. If no initial choice has been set, the initial choice for the device is used.
7. The default number of menu items and the initial value vary with each device.
8. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.

xample

```
PROGRAM tax_data
EXTERNAL SUB More_questions
DECLARE LONG marital_status,                                 &
        STRING form
OPEN "RT75" FOR GRAPHICS AS DEVICE #1
SET INITIAL CHOICE #1 , LIST                                 &
                                  ("Single",                 &
                                   "Divorced",               &
                                   "Widowed",                &
                                   "Married",                &
                                   "Other")                  &
                  : 5
```

```
!+
!Display menu, set a new initial choice & accept input
!-
LOCATE CHOICE #1 , INITIAL 5 : marital_status
SELECT marital_status
       CASE = 1%       !Single
                CALL More_questions
       CASE = 4%       !Married
               Form = "joint"
       CASE = 2% OR 3%  !Divorced or widowed
               Form = "long"
       CASE ELSE
              CALL More_questions
END SELECT
    .
    .
    .
END PROGRAM
```

Output



```
Single
Divorced
Widowed
Married
Other
```

ZK-4960-86

LOCATE POINT

A LOCATE POINT statement accepts a single point input by a user.

Format

```
              [ #dev-id                                               ]
LOCATE POINT  [ , UNIT int-exp1                               { : }   ]
              [ , AT x-coord1, y-coord1 [ USING TRAN int-exp2]        ]
              x-coord2, y-coord2 [ , int-var ]
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. *Int-exp2* must be an integer between 1 and 255.
3. If one or more optional clauses are included, one colon (:) is required before *x-coord2*.

Remarks

1. The world coordinates of the user-supplied input are assigned to the variables *x-coord2* and *y-coord2*.
2. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.

3. An initial point can be specified in the optional AT clause. VAX BASIC uses the current transformation for output to interpret the world coordinates of this point. You can specify an alternative transformation for interpreting this initial point in the optional USING TRAN clause. If an initial point is specified in the LOCATE POINT statement, this point overrides an initial point previously set with the SET INITIAL POINT statement.
4. The transformation with the highest priority is 1, unless a SET WINDOW, SET VIEWPORT, or SET TRANSFORMATION statement has been executed. See also the SET INPUT PRIORITY statement.
5. If multiple transformations are defined, the user-supplied point is evaluated using the transformation with the highest input priority of all transformations that contain the input point in their viewport. VAX BASIC assigns the actual number of the transformation used to optional *int-var*.
6. The initial point is indicated to the user with a device dependent prompt such as a tracking plus sign. The user can move the prompt with the arrow keys or a mouse to indicate a new point. To enter a selected point, the user presses RETURN or performs a similar activity The actual actions required are device dependent.
7. The program waits until the user indicates that input is complete. Program execution continues with the statement lexically after the LOCATE POINT statement.
8. Unlike points accepted by the GET POINT statement, points accepted by a LOCATE POINT statement within a picture are not affected by any transformation functions in the DRAW statement that invoked th picture. See the example for the MAT GET POINTS statement for an illustration of this difference.
9. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of thes methods of data entry is a different *unit*. The default unit value for *int-exp1* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as th VAX GKS documentation.

xample

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE boat(SINGLE,)
DECLARE LONG alt_tran,which_tran,                       &
        SINGLE user_x, user_y
!+
!Specify alternative transformation for initial point
!-
alt_tran = 2
!+
!Request coordinate pair from user & set initial point
!-
LOCATE POINT , AT 50,50 USING TRAN alt_tran : user_x, user_y , which_tran
!+
!Use input coordinates as starting point for a picture invocation
!-
DRAW boat(user_x,user_y)
    .
    .
    .
END
```

LOCATE STRING

The LOCATE STRING statement accepts a user's string input.

Format

```
                 [ #dev-id              {   }  ]
LOCATE STRING    [ , UNIT int-exp       { : }  ]  str-var
                 [ , INITIAL str-exp    {   }  ]
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. If one or more optional clauses are included, one colon (:) is require before *str-var*.

Remarks

1. The string supplied by the user is assigned to *str-var*.
2. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses default identification of #0. If an identification clause is included, th device specified must have been opened explicitly, or opened by VA BASIC as the default device.
3. The optional INITIAL clause allows you to specify an initial string fc the user. This initial string is displayed within the string echo area. The user can enter this initial string, or an alternative. The initial string specified in a LOCATE STRING statement overrides an initial string specified in a SET INITIAL STRING statement.

4. When no INITIAL clause is included, the initial string in the last SET INITIAL STRING statement is used. If no initial string has been set, the default string for the device is used (usually the null string).
5. VAX BASIC appends the user-supplied string to the initial string you specify in a LOCATE STRING or a SET INITIAL STRING statement. If the user accepts the initial string with a null response, *str-var* contains only the initial string. Depending on the device, it may be possible for a user to delete the initial string by pressing CTRL/U or the DELETE key. You can use the VAX BASIC string functions to locate the exact user input.
6. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.

xample

```
OPTION TYPE = EXPLICIT
DECLARE STRING init_string, user_string, response
init_string = "YES"
LOCATE STRING , INITIAL init_string : user_string
!+
!Check if user responded with a <CR>
!-
IF user_string = init_string
   THEN response = init_string
        GOTO print_it_out
   ELSE GOTO extract_string
END IF
extract_string:
  !+
  !Check if all of init_string has been deleted or not
  !-
  IF LEFT$(user_string,3) = LEFT$(init_string,3)
    THEN response = RIGHT$(user_string,4)
    ELSE response = user_string
  END IF
print_it_out:
  PRINT "User entered '" + response + "'"
END
```

LOCATE VALUE

The LOCATE VALUE statement accepts a numeric value input by a user.

Format

```
             [ #dev-id                              ]
             [ , UNIT int-exp                  {  } ]
LOCATE VALUE [ , RANGE real-exp1 TO real-exp2  { : } ] real-var
             [ , INITIAL real-exp3             {  } ]
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. *Real-exp1* must be less than *real-exp2* in the optional RANGE clause.
3. *Real-exp3* must be within the optional range when specified. If no RANGE clause is included, *real-exp3* must be within the default range
4. You can include one or more optional clauses.
5. A colon must precede *real-var* when one or more optional clauses are present.

emarks

1. The value input by the user is assigned to *real-var*.
2. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
3. The optional RANGE clause allows you to specify the extent of acceptable values to the user. Intermediate values are scaled proportionately.
4. The RANGE clause specifies the lowest and highest acceptable input. The user-supplied value and the initial value must be within this range. VAX BASIC signals the run-time error ILLINIVAL, "illegal initial value" (ERR = 284), when the initial value is not within the specified range.
5. When no RANGE clause is included, the range in the last SET INITIAL VALUE statement is used. If no initial values have been set, the default range is 0 to 1.
6. The optional INITIAL clause allows you to specify an initial value for the user. This initial value is displayed within the specified range in the echo area. The initial value and range specified in a LOCATE VALUE statement override the value and range specified in a SET INITIAL VALUE statement.
7. The program waits until the default response or an alternative value is entered at the terminal. To enter a selected value, the user presses RETURN or performs a similar activity. The actual actions are device dependent.
8. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
9. For an illustration of the VALUE echo and prompt, see the SET INITIAL VALUE statement.

Example

```
EXTERNAL SUB small_claims(SINGLE),            &
             night_court(SINGLE),             &
             probate
DECLARE SINGLE limit1, limit2, your_number
limit1 = 0
limit2 = 10
LOCATE VALUE , RANGE limit1 TO limit2         &
             , INITIAL 2.5                    &
             : your_number
SELECT your_number
       CASE < 2.5
            CALL small_claims(your_number)
       CASE = 2.5
            CALL night_court(your_number)
       CASE > 2.5
            CALL probate
END SELECT
   .
   .
   .
```

IAT GET POINTS

The MAT GET POINTS statement accepts one or more points input by a user.

ormat

```
                 [ #dev-id                                           {   } ]
MAT GET POINTS   [ , UNIT int-exp1                                   { : } ]
                 [ , COUNT int-var1                                  {   } ]
                 [ , AT x-coord, y-coord [ USING TRAN int-exp2 ]     {   } ]
               x-array, y-array [ , int-var2]
```

yntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. With the MAT statement, specify only the array name; do not use parentheses to indicate the whole array.
3. *X-array* and *y_array* must be one-dimensional arrays that contain only integer or real data types. Virtual arrays are not valid.
4. *Int-exp2* must be an integer between 1 and 255.
5. If one or more optional clauses are included, one colon (:) is required before *x-array*.

Remarks

1. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. When multiple transformations are defined, the user-supplied input is interpreted with the transformation with the highest priority that contains all the points input by the user. The world coordinates of the user-supplied points are assigned to *x-array* and *y-array*. VAX BASIC assigns the number of the transformation used to *int-var2*. See also the SET INPUT PRIORITY statement.
3. An initial point can be specified in the optional AT clause. VAX BASIC uses the current output transformation to interpret the world coordinates of this point for display on the user's screen. You can specify an alternative transformation for interpreting this initial point by including an optional USING TRAN clause. If an initial point is specified in a MAT GET POINTS statement, this point overrides an initial point previously set with a SET INITIAL MULTIPOINT or SET INITIAL POINT statement.
4. The optional COUNT clause allows you to find out how many points the user actually entered. If a user enters more points than the input arrays can contain, VAX BASIC discards the extra points and the user cannot retrieve these values; however, *int-var1* still retains the actual number entered. When an unknown number of points is input and you want to display these points, you need to examine *int-var1* before using this number in the MAT GRAPH or MAT PLOT statements you use for related output. The following example shows one way of ensuring that the COUNT clause is valid for the output statement.

Example

```
DECLARE LONG CONSTANT in_max = 12
DECLARE LONG how_many
DIM SINGLE in_x(1 TO in_max),in_y(1 TO in_max)
MAT GET POINTS , COUNT how_many : in_x, in_y
IF how_many > in_max
   THEN GRAPH TEXT AT 0,0.9: "You can only input"
        GRAPH TEXT AT 0,0.8: STR$(in_max) + " points. Your"
        GRAPH TEXT AT 0,0.7: "extra points have been discarded."
        how_many = in_max
END IF
MAT GRAPH POINTS , COUNT how_many : in_x, in_y
```

5. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp1* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
6. Points accepted with the MAT GET POINTS statement in a PICTURE subprogram are transformed through the inverse of any transformation functions on the DRAW statement that invoked the picture. Points accepted by the MAT LOCATE POINTS statement, however, are not affected by the transformation functions.

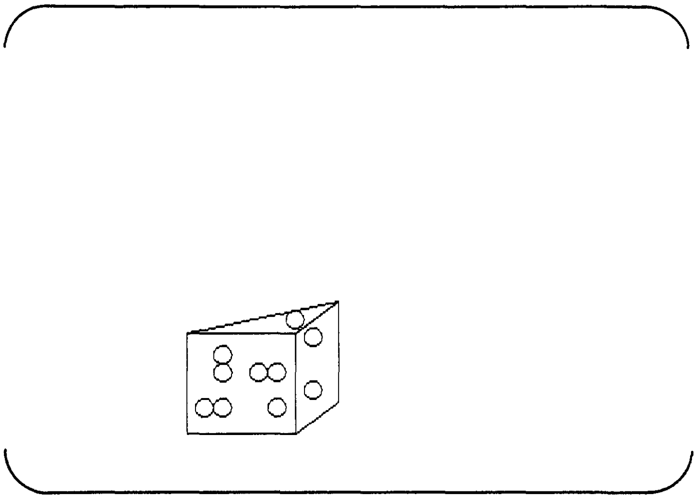
:xample

The following example illustrates the difference between the MAT GET POINTS and MAT LOCATE POINTS statements. The points accepted by the GET statement are scaled according to the transformation function in the DRAW statement; therefore, these points are displayed within the cheese outline. The points accepted by the LOCATE statement are not scaled along with the cheese outline; therefore, the points are not all displayed within the outline.

```
EXTERNAL PICTURE get_the_points                              &
               , locate_the_points

!Circles for the holes in the cheese
SET POINT STYLE 4
DRAW get_the_points WITH SCALE(1/2)
SLEEP 5%
CLEAR
```

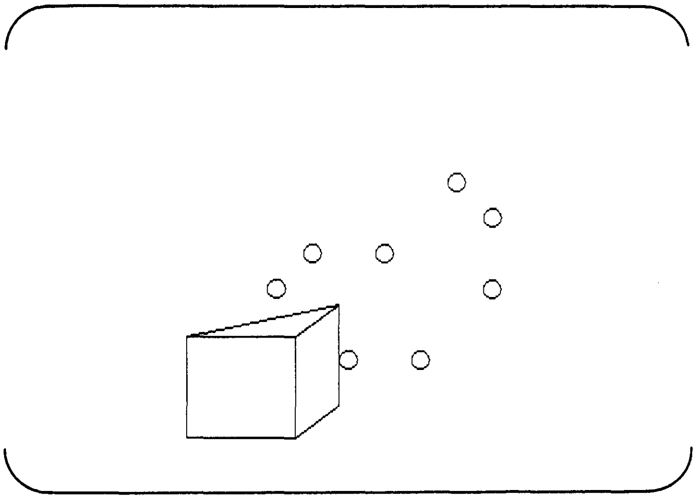
```
DRAW locate_the_points WITH SCALE(1/2)
END

PICTURE swiss_cheese
  DECLARE LONG counter
  DIM SINGLE outline_x(9),outline_y(9)
  DATA 0.25,0.6, 0.25,0.125, 0.75,0.125, 0.75,0.6, 0.25,0.6,      &
       0.95,0.75, 0.75,0.6, 0.95,0.75, 0.95,0.275, 0.75,0.125
  READ outline_x(counter),outline_y(counter) FOR counter = 0 TO 9
  MAT PLOT LINES outline_x, outline_y
END PICTURE

PICTURE get_the_points
   EXTERNAL PICTURE swiss_cheese
   DECLARE LONG CONSTANT input_max = 12
   DECLARE LONG how_many
   DIM SINGLE get_xholes(input_max),get_yholes(input_max)
   DRAW swiss_cheese
   MAT GET POINTS ,COUNT how_many : get_xholes, get_yholes
   IF how_many > (input_max + 1)
      THEN how_many = input_max + 1
   END IF
   MAT GRAPH POINTS , COUNT how_many : get_xholes, get_yholes
END PICTURE

PICTURE locate_the_points
   EXTERNAL PICTURE swiss_cheese
   DECLARE LONG CONSTANT input_max = 12
   DECLARE LONG how_many
   DIM SINGLE locate_xholes(input_max),locate_yholes(input_max)
   DRAW swiss_cheese
   MAT LOCATE POINTS , COUNT how_many : locate_xholes, locate_yholes
   IF how_many > (input_max + 1)
      THEN how_many = input_max + 1
   END IF
   MAT GRAPH POINTS , COUNT how_many : locate_xholes, locate_yholes
END PICTURE
```

Output from Get_the_points:



ZK-5232-86

Output from Locate_the_points:



ZK-5233-86

MAT GRAPH

The MAT GRAPH statement draws graphics output specified by the worl coordinate points contained in arrays. The output is drawn on all active devices. Values for graphics attributes such as color and style depend on the current settings.

Format

$$\textbf{MAT GRAPH} \quad \left\{ \begin{array}{l} \textbf{POINTS} \\ \textbf{LINES} \\ \textbf{AREA} \end{array} \right\} [\ ,\ \textbf{COUNT}\ \textit{int-exp}:\ ]\ \textit{x-array},\ \textit{y-array}$$

Syntax Rules

1. With the MAT statement, specify only the array name; do not use parentheses to indicate the whole array.
2. Both *x-array* and *y-array* must be one-dimensional arrays containing integer or real data types. Virtual arrays are not valid.
3. *Int-exp* cannot be greater than the number of elements in the coordinate arrays.
4. A MAT GRAPH POINTS statement must include the coordinates for at least one point with each coordinate in the appropriate array.
5. A MAT GRAPH LINES statement requires a minimum of two points.
6. A MAT GRAPH AREA statement requires a minimum of three coordi nate pairs.

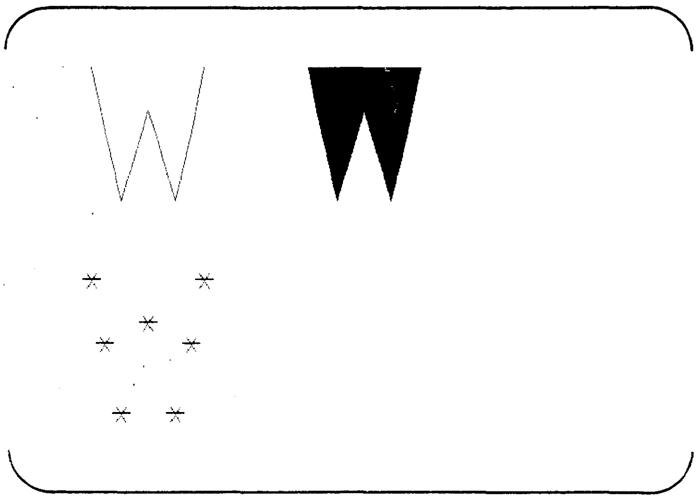
Remarks

1. Points specified in arrays must begin in the element with the lowest subscript. An optional COUNT clause allows you to display a specified number of the points described in the array elements. If no COUNT clause is included, all of the points specified in the arrays are displayed.
2. The sequence of objects drawn is determined by the order of the coordinates in the arrays in the MAT GRAPH statement; the first pair of coordinates supplied in both arrays designates the first point, the second pair designates the second point, and so on through the end of the arrays.
3. If MAT GRAPH LINES is used to draw a closed figure, the coordinates of the first point should be supplied again for the last point. For example, to draw a square with MAT GRAPH LINES, you must supply the four points of the square and then supply the first point again as the fifth point.
4. To draw closed areas with the MAT GRAPH AREA statement, you supply the same number of points as there are vertices in the polygon.
5. The coordinates of points in MAT GRAPH statements within a picture definition are not affected by the transformations specified in the DRAW statement that invoked the picture. To transform points within a picture definition, you must use the MAT PLOT statement. However, when you know that transformation functions will not be used, use GRAPH rather than PLOT statements, because they are more efficient.

Example

```
PROGRAM letter_w
  DECLARE LONG counter
  DIM SINGLE x_coords(6),y_coords(6)
  DATA 0.24,0.75, 0.3,0.45, 0.375,0.125, 0.5,0.55,        &
       0.625,0.125, 0.7,0.45, 0.76,0.75
  READ x_coords(counter),y_coords(counter) FOR counter = 0% TO 6%

  !For illustration, the same coords are used
  !in each of the following MAT statements
  SET VIEWPORT ,TRAN 1 :0,0.5,0,0.5
  MAT GRAPH POINTS x_coords, y_coords
  SET VIEWPORT ,TRAN 2 : 0,0.5,0.5,1
  MAT GRAPH LINES x_coords, y_coords

  SET VIEWPORT ,TRAN 3 : 0.5,1,0.5,1
  MAT GRAPH AREA x_coords, y_coords
END PROGRAM
```

Output



ZK 4982-86

MAT LOCATE POINTS

The MAT LOCATE POINTS statement accepts one or more points input by a user.

Format

```
                    [ #dev-id                                                ]
MAT LOCATE POINTS   [ , UNIT int-exp1                                  { : } ]
                    [ , COUNT int-var1                                       ]
                    [ , AT x-coord, y-coord [ USING TRAN int-exp2 ]          ]

          x-array, y-array [ , int-var2]
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. With the MAT statement, specify only the array name; do not use parentheses to indicate the whole array.
3. *X-array* and *y_array* must be one-dimensional arrays that contain only integer or real data types. Virtual arrays are not valid.
4. *Int-exp2* must be an integer between 1 and 255.
5. If one or more optional clauses are included, one colon (:) is required before *x-array*.

Remarks

1. The optional *dev-id* identifies the device from which you want to accept input. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. When multiple transformations are defined, the user-supplied input is interpreted with the transformation with the highest priority that contains all the points input by the user. The world coordinates of the user-supplied points are assigned to *x-array* and *y-array*. VAX BASIC assigns the number of the transformation used to *int-var2*. See also the SET INPUT PRIORITY statement.
3. An initial point can be specified in the optional AT clause. VAX BASIC uses the current output transformation to interpret the world coordinates of this point for display on the user's screen. You can specify an alternative transformation for interpreting this initial point by including an optional USING TRAN clause. If an initial point is specified in a MAT LOCATE POINTS statement, this point overrides an initial point previously set with a SET INITIAL MULTIPOINT or SET INITIAL POINT statement.
4. The optional COUNT clause allows you to find out how many points the user actually entered. If a user enters more points than the input arrays can contain, VAX BASIC discards the extra points and the user cannot retrieve these values; however, *int-var1* still retains the actual number entered. When an unknown number of points are input and you want to display these points, you need to examine *int-var1* before using this number in the MAT GRAPH or MAT PLOT statements you use for related output. The example at the end of this statement shows one way of ensuring that the COUNT clause is valid for the output statement.
5. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp1* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.

6. Points accepted with the MAT LOCATE POINTS statement in a PICTURE subprogram are not affected by any transformation functions on the DRAW statement that invoked the picture. Points accepted by the MAT GET POINTS statement, however, are transformed by the inverse of the transformation functions. See the MAT GET POINTS statement for an example illustrating this difference.

Example

```
DECLARE LONG CONSTANT in_max = 12
DECLARE LONG how_many
DIM in_x(in_max),in_y(in_max)
MAT LOCATE POINTS , COUNT how_many : in_x, in_y
IF how_many > in_max
   THEN GRAPH TEXT AT 0,0.9: "You can only input"
        GRAPH TEXT AT 0,0.8: STR$(in_max) + " points. Your"
        GRAPH TEXT AT 0,0.7: "extra points have been discarded."
        how_many = in_max
END IF
SET POINT COLOR 2
MAT GRAPH POINTS , COUNT how_many : in_x, in_y
```

MAT PLOT

The MAT PLOT statement draws graphics output specified by the world coordinate points contained in arrays. Points drawn with the MAT PLOT statement within a picture are affected by the transformation functions included in the DRAW statement that invokes the picture. The output is drawn on all active devices. Values for graphics attributes such as color and style depend on the current settings.

Format

$$\textbf{MAT PLOT} \begin{Bmatrix} \textbf{POINTS} \\ \textbf{LINES} \\ \textbf{AREA} \end{Bmatrix} [\textbf{, COUNT}\ \textit{int-exp}:\ ]\ \textit{x-array, y-array}$$

Syntax Rules

1. With the MAT statement, specify only the array name; do not use parentheses to indicate the whole array.
2. Both *x-array* and *y-array* must be one-dimensional arrays containing integer or real data types. Virtual arrays are not valid.
3. *Int-exp* cannot be greater than the number of elements in the coordinate arrays.
4. A MAT PLOT POINTS statement must include the coordinates for at least one point with each coordinate in the appropriate array.
5. A MAT PLOT LINES statement requires a minimum of two points.
6. A MAT PLOT AREA statement requires a minimum of three coordinate pairs.

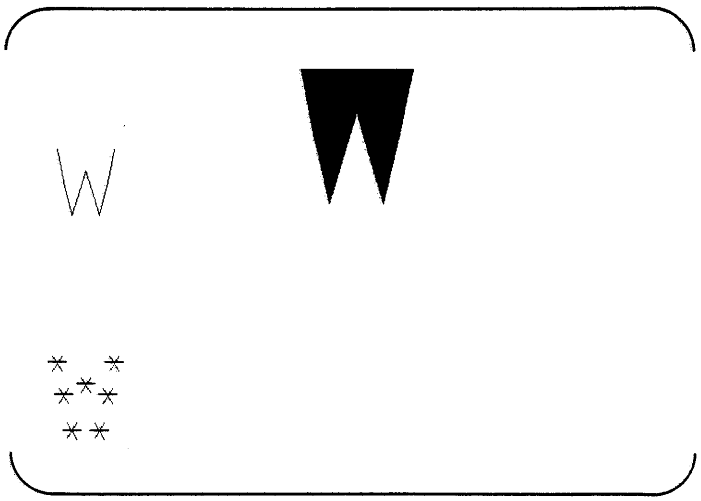
Remarks

1. Points specified in arrays must begin in the element with the lowest subscript. An optional COUNT clause allows you to display a specified number of the points described in the arrays. If no COUNT clause is included, all of the points specified in the arrays are displayed.
2. The sequence of objects drawn is determined by the order of the coordinates in the arrays in the MAT PLOT statement; the first pair of coordinates supplied in both arrays designates the first point, the second pair designates the second point, and so on through the end of the arrays.
3. If MAT PLOT LINES is used to draw a closed figure, the coordinates of the first point should be supplied again for the last point. For example, to draw a square with MAT PLOT LINES, you must supply the four points of the square and then supply the first point again as the fifth point.
4. To draw closed areas with the MAT PLOT AREA statement, you supply the same number of points as there are vertices in the polygon.
5. The coordinates of points in MAT PLOT statements within a picture definition are affected by the transformations in the DRAW statement that invokes the picture. However, when you know that transformation functions will not be used, use GRAPH rather than PLOT statements, because they are more efficient.

Example

```
PROGRAM letter_w
 EXTERNAL PICTURE area_w(SINGLE DIM (), SINGLE DIM ()),              &
                  lines_w(SINGLE DIM (), SINGLE DIM ()),             &
                  points_w(SINGLE DIM (), SINGLE DIM ())
 DECLARE LONG counter
 DIM SINGLE x_coords(6),y_coords(6)
 SET VIEWPORT ,TRAN 1 :0,0.5,0,0.5
 SET VIEWPORT ,TRAN 2 : 0,0.5,0.5,1
 SET VIEWPORT ,TRAN 3 : 0.5,1,0.5,1

 DATA 0.24,0.75, 0.3,0.45, 0.375,0.125, 0.5,0.55,                    &
      0.625,0.125, 0.7,0.45, 0.76,0.75
 READ x_coords(counter),y_coords(counter) FOR counter = 0% TO 6%

 SET TRANSFORMATION 1
 DRAW points_w(x_coords(),y_coords()) WITH SCALE(0.5)
 SET TRANSFORMATION 2
 DRAW lines_w(x_coords(),y_coords())  WITH SCALE(0.5)

 SET TRANSFORMATION 3
 SET AREA STYLE "SOLID"
 DRAW area_w(x_coords(),y_coords()) WITH SCALE(0.5)
END PROGRAM

PICTURE points_w(SINGLE x_coords(),y_coords())
 MAT PLOT POINTS x_coords, y_coords
END PICTURE

PICTURE lines_w(SINGLE x_coords(),y_coords())
  MAT PLOT LINES x_coords, y_coords
END PICTURE

!+
!This picture uses MAT GRAPH rather than MAT PLOT
!See the output of area_w compared to points_w and lines_w
!-
PICTURE area_w(SINGLE x_coords(),y_coords())
  MAT GRAPH AREA x_coords, y_coords
END PICTURE
```

Output



ZK-4968-86

PEN . . . FOR GRAPHICS

The OPEN . . . FOR GRAPHICS statement opens and activates the specified device for graphics input or output.

ormat

OPEN { *dev-name* / *file-spec* } **FOR GRAPHICS AS** *[* **DEVICE** *]* **#***dev-id* *[* **, TYPE** *dev-type]*

yntax Rules

1. *Dev-name* can be a string expression or a quoted string representing a logical name for a device.
2. *File-spec* must be a valid metafile.
3. You must supply the number sign (#) with *dev-id*.
4. *Dev-type* must be a valid integer expression representing a device supported by VAX GKS and must correspond to the actual device (or metafile) specified in *dev-name*. See Table 9–1 in this section for a list of supported device types.

emarks

1. If only the default device is used, you need not use the OPEN . . . FOR GRAPHICS statement. When graphics output statements are executed and no device has been opened, VAX BASIC opens the default device. The default device is opened with the equivalent of the following statement:

```
OPEN "" FOR GRAPHICS AS DEVICE #0
```

VAX BASIC opens the default device as a monochrome VT240 (devic type 14), unless an alternative device type has been assigned to the logical name GKS$WSTYPE. To find out if GKS$WSTYPE has been assigned system-wide, use the following command:

```
$ SHOW LOGICAL GKS$WSTYPE
```

If a device type has been assigned to GKS$WSTYPE, VAX BASIC use this type as the default device; however, if no device type has been assigned, VAX BASIC assumes the device is a monochrome VT240.

To assign a specific device type as your own default, use the DCL command ASSIGN. The following command informs VAX GKS that you are using a VT240 terminal with the color option (device type 13). This assignment overrides any system value assigned to GKS$WSTYPE.

```
$ ASSIGN 13 GKS$WSTYPE
```

2. An OPEN . . . FOR GRAPHICS statement implicitly activates the identified device.
3. Table 9–1 in this section lists the possible values for *dev-type*. This list is complete up to the print date on this manual. For up-to-date information, consult the VAX GKS documentation.
4. When an OPEN . . . FOR GRAPHICS statement specifies a device that is already open, VAX BASIC closes the device, then reopens it with the specifications in the most recent OPEN . . . FOR GRAPHIC statement.
5. The *dev-id* specified in an OPEN . . . FOR GRAPHICS statement is not related to the channels specified in the OPEN statement for file I/O. For example, you could open a file on channel #1 and open a graphics device as #1. Channel #1 and device #1 are independent o each other.
6. If you open a device that has color index values that are not contigu- ous, VAX BASIC issues an error message. Results from the ASK MA COLOR statement are unpredictable on such a device.
7. For information on opening a metafile, see the GRAPH METAFILE statement.
8. Information that is specific to VAXstations and to VT125 and VT240 terminals is listed in Appendix B.

Table 9–1: Supported Device Types

Device Type	Supported Device[1]
0	Default device type
2	Output metafile
3	Input metafile
10	VT125 output only
11	VT125 with color
12	VT125 black and white
13	VT240 with color
14	VT240 black and white
15	LCP01 printer
31	LA34 with graphics option LA100 LA210
32	LA50 with 2:1 aspect ratio
33	LA12
41	VAXstation I VAXstation II monochrome VAXstation II/GPX color
51	LVP16 color plotter (8–1/2 by 11 paper)
52	LVP16 (11 by 17 paper)
70	Tektronix 4014 output only
72	Tektronix 4014

[1]It is possible that the device you use is not on this list but has been adapted to take advantage of VAX GKS. If this is the case, you should refer to both the hardware and the VAX GKS documentation and use caution in all device-specific clauses in your applications. This list was complete when this manual went to print; see the VAX GKS documentation for the latest information about supported devices.

OPEN . . . FOR GRAPHICS

Example

```
!+
!Open a VAXstation II
!-
OPEN "VTA24:" FOR GRAPHICS AS DEVICE #1 , TYPE 41
!+
!Open a terminal as the default device type
!-
OPEN "my_term" FOR GRAPHICS AS DEVICE #2
!+
!Open an output metafile
!-
OPEN "meta.pic" FOR GRAPHICS AS DEVICE #3 , TYPE 2
    .
    .
    .
```

ICTURE

A PICTURE subprogram defines a graphics display and can be invoked with the DRAW statement. The PICTURE statement marks the start of a picture definition.

ormat

```
PICTURE   pic-name [(param-list)]

          statement
          [statement]
              .
              .
              .

END PICTURE
```

yntax Rules

1. *Pic-name* must be a valid VAX BASIC identifier and must not be the same as an identifier for a SUB or FUNCTION subprogram or a PROGRAM unit.
2. Parameters in the *param-list* must agree in number and data type with the parameters in a picture invocation (a DRAW statement).

Remarks

1. A definition of a picture must be delimited by the PICTURE and END PICTURE statements.
2. Statements within a picture can invoke other procedures.
3. Pictures can be recursive; a picture can include a DRAW statement that invokes itself. These DRAW statements can be either within the picture or within a called subprogram.
4. Statements that affect the boundaries of windows and viewports as well as statements that affect the clipping status are not valid within a picture. Errors are signaled if these statements are lexically within a picture definition, or if the statements are executed in a subprogram called from a picture. Invalid statements include the following:
   - SET CLIP
   - SET DEVICE VIEWPORT
   - SET DEVICE WINDOW
   - SET INPUT PRIORITY
   - SET TRANSFORMATION
   - SET VIEWPORT
   - SET WINDOW
5. Pictures are invoked with the DRAW statement and can be rotated, scaled, shifted, tilted, and otherwise modified by transformation functions included in the DRAW statement. For more information about invoking pictures, see the DRAW statement.
6. Transformation functions in DRAW statements within a picture are cumulative. That is, when a second or subsequent picture is invoked within a picture definition, the transformations from all of the relevan DRAW statements are applied to the points generated by the second o subsequent picture. For example, in the following picture, the picture *nest2* is invoked with a DRAW statement equivalent to DRAW WITH SCALE(2,1) * SHIFT(50,0).

Example

```
PICTURE nest 1
EXTERNAL nest2
   .
   .
   .
   DRAW nest2 WITH SCALE(2,1)
END PICTURE
!+
!Invoke picture nest1
!-
DRAW nest1 WITH SHIFT(50,0)
```

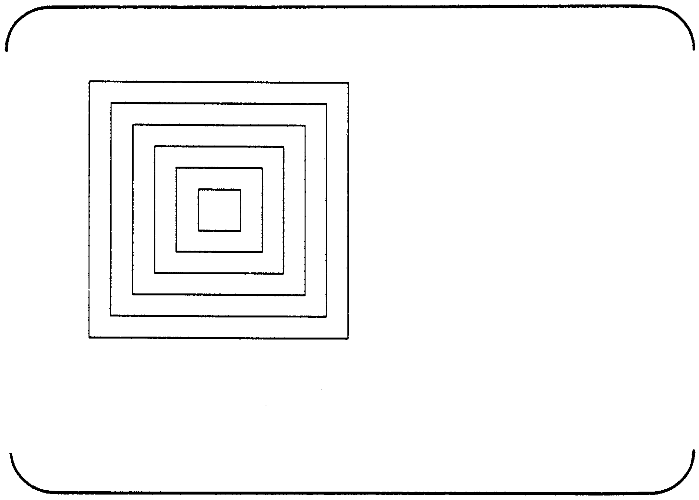
7. When graphics output statements using GRAPH or MAT GRAPH are contained within a picture, the points displayed are not affected by the transformation functions in the DRAW statement that invokes the picture. Output displayed with the PLOT and MAT PLOT statements, on the other hand, is affected by the transformation functions. See the GRAPH and PLOT statements for more information.
8. When graphics input statements using LOCATE are contained within a picture, the points entered are not affected by the transformation functions in the DRAW statement that invokes the picture. Input points accepted by the GET statement are affected by the transformation functions. For more information, see the LOCATE and GET POINTS statements, as well as the MAT GET POINTS statement.
9. For more examples of pictures, see Chapter 6 in this manual. See Chapter 7 for information about accepting input within picture definitions.

:xample

```
EXTERNAL PICTURE square(LONG)
DECLARE LONG width
SET WINDOW 0,100,0,100
DRAW square(width) FOR width = 60 TO 10 STEP -10
END
```

```
PICTURE square(LONG width)
  DECLARE SINGLE x_top_left,y_top_left,         &
                 x_top_right,y_top_right,       &
                 x_bottom_right,y_bottom_right, &
                 x_bottom_left,y_bottom_left
  x_top_left = 50 - width/2
  y_top_left = 50 + width/2
  x_top_right = y_top_left
  y_top_right = y_top_left
  x_bottom_right = x_top_right
  y_bottom_right = x_top_left
  x_bottom_left = x_top_left
  y_bottom_left = y_bottom_right

  PLOT LINES : x_top_left, y_top_left;         &
               x_top_right,y_top_right;        &
               x_bottom_right,y_bottom_right;  &
               x_bottom_left,y_bottom_left;    &
               x_top_left,y_top_left
END PICTURE
```

Output



ZK-4962-86

OT

The PLOT statement draws the specified graphics object on all active devices. Values for graphics attributes such as color and style use the current settings. Graphics objects displayed with PLOT from within a picture are affected by the transformation functions specified on the DRAW statement that invokes the picture.

mat

$$\text{LOT} \left\{ \begin{array}{l} \textbf{POINTS} \\ \textbf{LINES} \\ \textbf{AREA} \end{array} \right\} [\ :\ ]\ \textit{x-coord}\ ,\ \textit{y-coord}\ [\ ;\ \textit{x-coord},\ \textit{y-coord}\ ]...$$

ıtax Rules

1. A PLOT POINTS statement must include the coordinates for at least one point.
2. A PLOT AREA statement must include the coordinates for at least three points.
3. The last point specified in a PLOT LINES statement can be followed by an optional semicolon (see the remarks below).

ıarks

1. The sequence of drawing the objects is determined by the order of the coordinate pairs presented in the PLOT statement; the first coordinate pair designates the first point, the second pair designates the second point, and so on through the end of the coordinate pairs.

2. A "beam of light" can be left on if you add a semicolon after the last point specified in a PLOT LINES statement. When this beam (light is left on, a line will be drawn from this point to the next poi specified in a subsequent PLOT LINES statement. For instance, th first PLOT LINES statement in the following example leaves the be of light on and a line is drawn between points 50,50 and 75,75. Tl beam of light is then turned off because no semicolon is present af the second PLOT LINES statement.

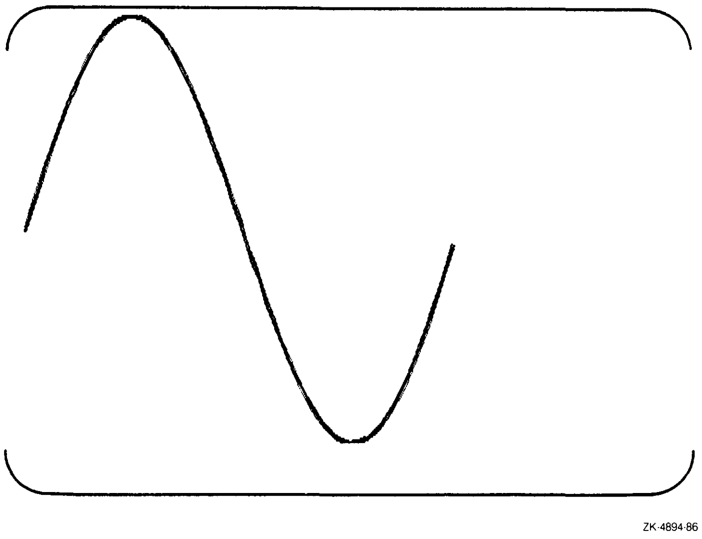
```
PLOT LINES 50,50; !beam is left on
      .
      .
      .
PLOT LINES 75,75  !beam is turned off
```

3. The output from PLOT LINES statements that leave the beam on is not actually displayed until the beam is switched off with the execution of any graphics output statement other than PLOT LINE with a semicolon. If no further graphics output statement is execut the output is displayed when program execution terminates.
4. A PLOT LINES statement with no points supplied switches off the beam of light.
5. If PLOT LINES is used to draw a closed figure, the coordinates of t first point should be supplied again for the last point. For example to draw a square with PLOT LINES, you must supply the four poi of the square and then supply the first point again as the fifth poir These points can be supplied one at a time (such as in a loop), or (after the other in the same statement.
6. To draw closed areas with the PLOT AREA statement, you supply same number of points as there are vertices in the polygon.
7. The coordinates of points in PLOT statements within a picture defi tion are affected by the transformations in the DRAW statement th invokes the picture. However, when you know that transformatio functions will not be used, use GRAPH rather than PLOT statemer because they are more efficient.

mple

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE counter
SET WINDOW 0,2*PI,-1,1
SET LINE SIZE 5
FOR counter = 0 TO 2*PI STEP 0.1
  !Supply points 1 at a time
  !leave beam on to connect to next point
  PLOT LINES counter, SIN(counter);
NEXT counter
!+
!Complete the curve
!-
PLOT LINES 2 * PI,SIN(2 * PI)
END
```

Output



RESTORE GRAPHICS

The RESTORE GRAPHICS statement clears the current values for all attributes and input types, and restores the default values as at the star program execution.

Format

RESTORE GRAPHICS

Syntax Rules

None.

Remarks

1. Executing a RESTORE GRAPHICS statement closes all open device
2. When a RESTORE GRAPHICS statement is executed, all values for output attributes, characteristics of input types, and any device defaults are restored to their values at the start of program executic

Example

```
Declarations:
   .
   .
   .
trapper = CTRLC
WHEN ERROR USE cntrc_trap
 Start_up:
  OPEN "RT23" FOR GRAPHICS AS DEVICE #3
   .
   .
   .
END WHEN
```

```
HANDLER cntrc_trap
  !Test if user entered CTRL/C
  IF ERR = 28%
     !Restore original attributes and defaults
     THEN RESTORE GRAPHICS
          CONTINUE
  END IF
   .
   .
   .
END HANDLER
```

ROTATE

When used in a DRAW statement, the ROTATE function rotates points specified in a PICTURE subprogram. ROTATE can also be used in the MAT statement to create a new matrix.

Format

In the DRAW statement

DRAW *pic-name[(param-list)]* **WITH ROTATE***(angle) [* matrix2]...*

In the MAT statement

MAT *matrix1* = **ROTATE***(angle) [* matrix2]...*

Syntax Rules

1. *Angle* can be in radians or degrees, depending on the option you specify with the OPTION ANGLE statement. The default is radian
2. *Matrix1* and *matrix2* must be two-dimensional numeric arrays that zero based with upper bounds of 4 in both directions. VAX BASIC signals a compile-time error when the compiler detects a nonzero-based matrix; otherwise, a run-time error is signaled. Packed decin arrays are invalid.
3. *Matrix2* can also be one of the valid transformation functions from DRAW statement, including the following:
   - SCALE
   - SHEAR
   - SHIFT
   - TRANSFORM

marks

1. ROTATE can be used only on the right-hand side of a MAT statement or as a transformation function in the WITH clause of the DRAW statement.
2. Like other transformation functions, the ROTATE function affects coordinates displayed with PLOT and MAT PLOT statements within pictures. Similarly, input points accepted with GET and MAT GET statements within a picture are affected by the inverse of a ROTATE function specified on the DRAW statement that invokes the picture.
3. Points are rotated about the point of origin 0,0. A positive argument rotates the point x,y in a counterclockwise direction about the point of origin of the world coordinates. A negative value rotates the point in a clockwise direction. You can alter the point of origin by setting the window such that the point 0,0 is at the center of the window.
4. For an example and more information, see the DRAW statement in this chapter.

SCALE

When used in the DRAW statement, the SCALE function applies a scal factor to the coordinates of points specified in a PICTURE subprogram. SCALE can also be used in the MAT statement to create a new matrix.

Format

In the DRAW Statement

DRAW *pic-name[(param-list)]* **WITH SCALE***(real-exp1 [, real-exp2]) [* matrix2].*

In the MAT Statement

MAT *matrix1* = **SCALE***(real-exp1 [, real-exp2]) [* matrix2]...*

Syntax Rules

1. *Real-exp1* indicates the scale factor that is applied to the x-coordina
2. *Real-exp2* indicates the scale factor that is applied to the y-coordina
3. *Matrix1* and *matrix2* must be two-dimensional numeric arrays that zero-based with upper bounds of 4 in both directions. VAX BASIC signals a compile-time error when the compiler detects a nonzero-based matrix; otherwise, a run-time error is signaled. Packed decin arrays are invalid.
4. *Matrix2* can also be one of the valid transformation functions from DRAW statement, including the following:
   - ROTATE
   - SHEAR
   - SHIFT
   - TRANSFORM

emarks

1. SCALE can be used only on the right-hand side of a MAT statement or as a transformation function in the WITH clause of the DRAW statement.
2. When *real-exp2* is not included, the scale factor for *real-exp1* is applied to both the x- and y-coordinates.
3. Like other transformation functions, the SCALE function affects coordinates displayed with PLOT and MAT PLOT statements within pictures. Similarly, input points accepted with GET and MAT GET statements within a picture definition are affected by the inverse of a SCALE function specified on the DRAW statement that invokes the picture.
4. For an example and more information, see the DRAW statement in this chapter.

SET AREA COLOR

The SET AREA COLOR statement allows you to specify the color of the area fill.

Format

SET AREA COLOR *int-exp*

Syntax Rules

Int-exp must be a valid color value for the all devices in use, and must be greater than zero.

Remarks

1. Changing the area color affects the subsequent display of areas on all active devices with color capabilities. You cannot specify a particular device with this statement.
2. Changing color values may affect the shading on some black and white devices.
3. The actual colors displayed are device dependent. If you set the color to an index value that is undefined for a particular device, results are unpredictable.

4. On VT125 and VT240 terminals with the color option, the default values for *int-exp* represent the following colors:

Index	Color
0	Black
1	Green
2	Red
3	Blue

5. See also the SET COLOR MIX statement for related information.

:xample

```
PROGRAM samples
DECLARE LONG CONSTANT  green = 1          &
                      ,red = 2            &
                      ,blue =  3
  SET LINE COLOR green
  GRAPH LINES 0.05,1; 0.05,0
  SET POINT COLOR blue
  GRAPH POINTS 0.1,0.7; 0.1,0.3
  SET AREA COLOR red
  GRAPH AREA 0.1,0.5; 0.5,0.8;            &
             0.35,0.5; 0.5,0.2
  SET TEXT COLOR green
  GRAPH TEXT AT 0.6,0.5 : "Color me simple"
END PROGRAM
```

Output

ET AREA STYLE

The SET AREA STYLE statement lets you specify your choice of fill style for an area.

ormat

SET AREA STYLE *str-exp*

yntax Rules

The area style value can be one of the following four string values (in lower- or uppercase):

- HOLLOW
- SOLID
- PATTERN
- HATCH

emarks

1. The initial area style value is solid.
2. When the area style is hollow, the following two output statements produce the same results (a triangle with the outline displayed).

   Example

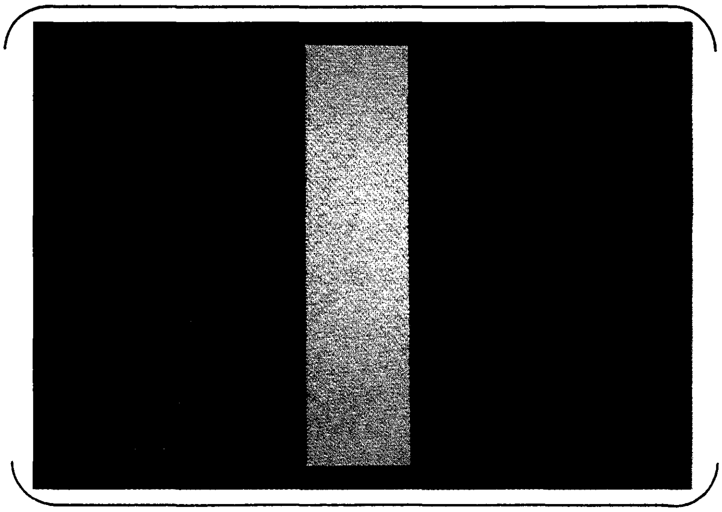
```
SET AREA STYLE "Hollow"
GRAPH AREA : 0.2,0.2;                    &
             0.4,0.8;                    &
             0.6,0.4
GRAPH LINES : 0.2,0.2;                   &
              0.4,0.8;                   &
              0.6,0.4;                   &
              0.2,0.2
```

3. The pattern and hatch styles both have indices that can be set with the SET AREA STYLE INDEX statement.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE sample
SET AREA STYLE "SOLID"
DRAW sample
SET AREA STYLE "HOLLOW"
DRAW sample WITH SHIFT(0.25,0)
SET AREA STYLE "PATTERN"
DRAW sample WITH SHIFT(0.5,0)
SET AREA STYLE "HATCH"
DRAW sample WITH SHIFT(0.75,0)
END

PICTURE sample
  DECLARE SINGLE x_sample(3), y_sample(3)
  x_sample(0) = 0.0
  x_sample(1) = 0.25
  x_sample(2) = 0.25
  x_sample(3) = 0.0
  y_sample(0) = 1.0
  y_sample(1) = 1.0
  y_sample(2) = 0.0
  y_sample(3) = 0.0
  MAT PLOT AREA x_sample, y_sample
END PICTURE
```

Output



SET AREA STYLE INDEX

The SET AREA STYLE INDEX statement lets you specify your choice of index for hatch or pattern area fill styles.

Format

SET AREA STYLE INDEX *int-exp*

Syntax Rules

None.

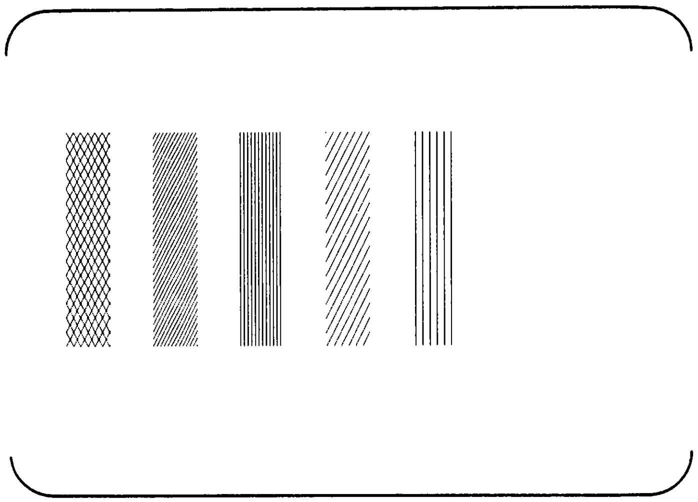
Remarks

1. The area style index value specifies one of the various hatch or pattern styles. The initial value for the index is 1. There are no indices for hollow or solid area styles. The indices available for VT125 and VT240 terminals are listed in Chapter 3.
2. Values for the AREA STYLE INDEX are device dependent. If you set the index to a value that is undefined for a device, results are unpredictable.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG variety
EXTERNAL PICTURE sample
SET AREA STYLE "hatch"
FOR variety = 1 TO 10 STEP 2
    SET AREA STYLE INDEX variety
    DRAW sample WITH SHIFT (0.1 * variety,0)
NEXT variety
END
```

Output



ZK-4955-86

SET CHOICE ECHO AREA

The SET CHOICE ECHO AREA statement lets you specify the boundaries of the CHOICE echo area.

Format

$$\textbf{SET CHOICE ECHO AREA} \left[\begin{array}{l} \#\textit{dev-id} \\ \textbf{, UNIT } \textit{int-exp} \end{array} \left\{ : \right\} \right]$$

$$\textit{real-exp1, real-exp2, real-exp3, real-exp4}$$

Syntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively.
2. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
3. If one or more optional clauses are included, one colon (:) is required before *real-exp1*.

Remarks

1. The echo area is the portion of your screen where the prompt appears and where input can be supplied by a user.
2. The optional *dev-id* identifies the device for which you want to set the echo area. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly or opened by VAX BASIC as the default device.

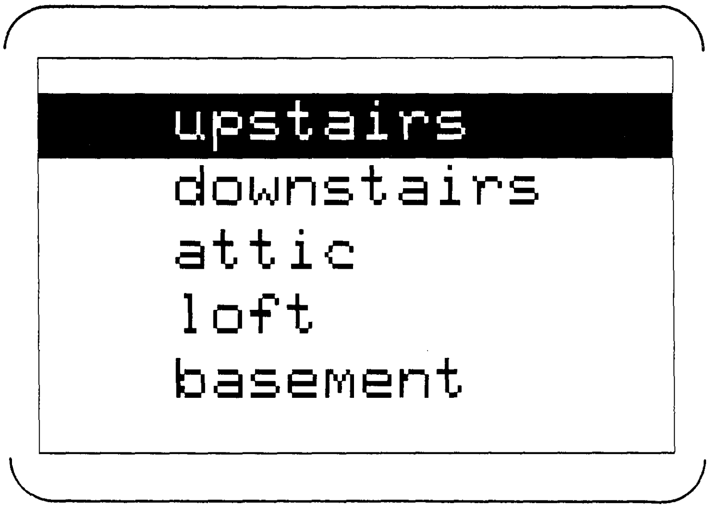
3. Boundaries must be specified in device coordinates. To determine the possible values on a particular device, use the ASK DEVICE SIZE statement. To determine the current boundaries, use the ASK CHOICE ECHO AREA statement. Appendix B lists the boundaries for VAXstations and VT125 and VT240 terminals.
4. The default echo area for CHOICE input is device dependent. The default area is in effect until a SET CHOICE ECHO AREA statement is executed.
5. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit.* The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
6. Be aware that the text height cannot be reduced lower than the smallest text height available for the device. Therefore, you should ensure that the CHOICE echo area is large enough to list the menu items you specify.
7. Subsequent requests for CHOICE input use the echo boundaries you supply until another SET CHOICE ECHO AREA statement is executed.

SET CHOICE ECHO AREA

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE left_1,right_1,bottom,top,        &
        LONG which_floor
OPEN "office-term" FOR GRAPHICS AS DEVICE #1
ASK CHOICE ECHO AREA #1 : left_1,right_1,bottom,top
!+
!Set the echo area to the full screen
!-
SET CHOICE ECHO AREA #1 : 0                       &
                          ,right_1                &
                          ,bottom                 &
                          ,top
SET INITIAL CHOICE , LIST ("upstairs",        &
                          "downstairs",       &
                          "attic",            &
                          "loft",             &
                          "basement")         &
                    :1
LOCATE CHOICE #1 : which_floor
    .
    .
    .
```

Output



ZK-5533-86

;ET CLIP

The SET CLIP statement lets you change the status of the clipping rectangle in the world viewport.

ormat

SET CLIP *str-exp*

yntax Rules

Values for *str-exp* can be equivalent to "ON" or "OFF" (in lower- or uppercase).

emarks

1. Clipping is enabled at the start of program execution.
2. When clipping is enabled with the SET CLIP "ON" statement, graphics objects with world coordinate values exceeding the limits of the world window are not displayed.
3. When clipping is disabled with the SET CLIP "OFF" statement, graphics objects with world coordinate values that exceed the limits of the world window are displayed providing that the points are also within the device window.
4. When clipping is enabled, STRING precision clips a text string at the world viewport boundary. No text string starts beyond the world viewport boundary.
5. When clipping is enabled, CHAR precision clips a text string by character at the world viewport. No character extends beyond the world viewport boundary. Some devices display part of a character that spans the world viewport boundary.

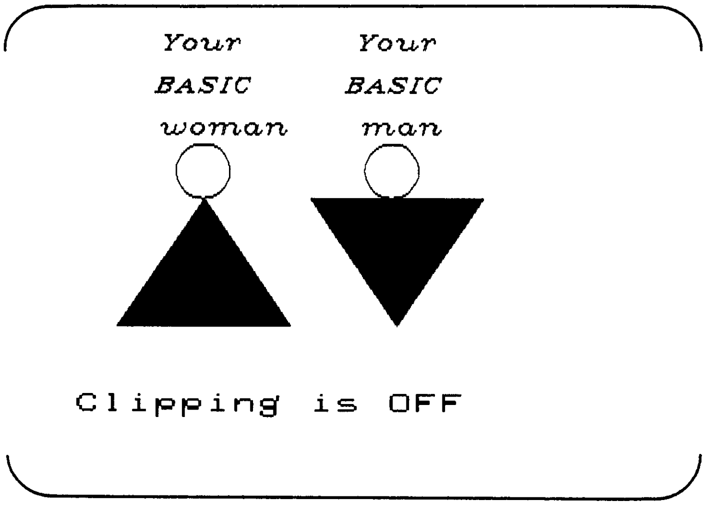
6. When clipping is enabled, STROKE precision clips text precisely at th
   world viewport boundary. For example, if only half a character falls
   inside the world viewport boundary, only this half is displayed on th
   screen.
7. The SET CLIP statement is invalid within a picture.

Example

```
PROGRAM demo_clip
  EXTERNAL PICTURE BASIC_people
  SET WINDOW 0,1,0,0.74
  SET VIEWPORT 0,1,0,0.74
  Screen1:
    SET CLIP "Off"
    DRAW BASIC_people
    SET TEXT FONT -1, "CHAR"
    GRAPH TEXT AT 0,0.1 : "Clipping is OFF"
    SLEEP 4%
    CLEAR
  Screen2:
    SET CLIP "On"
    DRAW BASIC_people
    SET TEXT FONT -1, "CHAR"
    GRAPH TEXT AT 0,0.1 : "Clipping is ON"
END PROGRAM

PICTURE BASIC_people
  EXTERNAL PICTURE circle
  DRAW circle WITH SCALE(0.125,0.125) * SHIFT(0.235,0.6)
  GRAPH AREA 0.3,0.6; 0.5,0.3; 0.1,0.3
  SET TEXT FONT -8, "STROKE"
  SET TEXT HEIGHT 0.05
  GRAPH TEXT AT 0.2,0.95 :"Your"
  GRAPH TEXT AT 0.185,0.85 :"BASIC"
  GRAPH TEXT AT 0.195,0.75 :"woman"
  DRAW circle WITH SCALE(0.125,0.125) * SHIFT(0.675,0.6)
  GRAPH AREA 0.55,0.6; 0.95,0.6; 0.75,0.3
  GRAPH TEXT AT 0.655,0.95 :"Your"
  GRAPH TEXT AT 0.628,0.85 :"BASIC"
  GRAPH TEXT AT 0.665,0.75 :"man"
END PICTURE
```

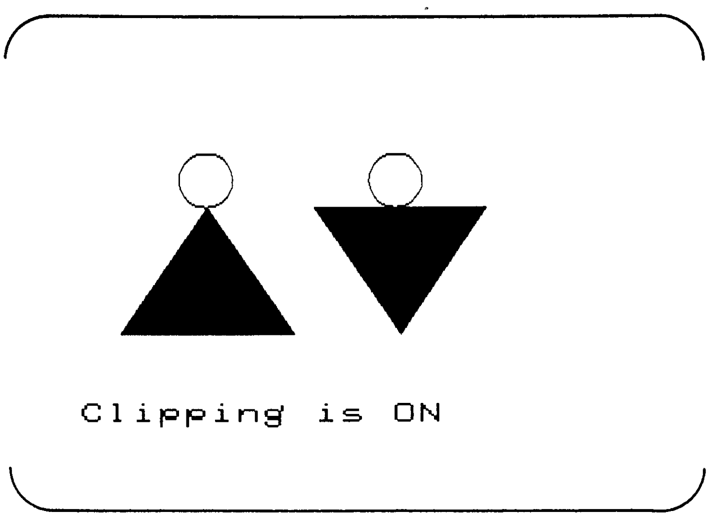
```
PICTURE circle
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE CONSTANT radius = 0.5
  DECLARE LONG CONSTANT npoints = 40
  DECLARE SINGLE angle,increment,x,y
  DIM SINGLE xs(npoints), ys(npoints)
  increment = 2*pi/npoints
  angle = 0
  FOR loop_count = 0 TO npoints - 1
     x = COS( angle ) * radius + 0.5
     y = SIN( angle ) * radius + 0.5
     xs(loop_count) = x
     ys(loop_count) = y
     angle = angle + increment
  NEXT loop_count
  xs(npoints) = xs(0)
  ys(npoints) = ys(0)
  MAT PLOT LINES xs, ys
END PICTURE
```

Output Screen 1:



ZK-5519-86

Output Screen 2:



ZK-5520-86

;ET COLOR MIX

The SET COLOR MIX statement allows you to specify the intensities for red, green, and blue for a particular color index.

ormat

SET COLOR MIX *[#dev-id]* **, INDEX** *int-exp* : *real-exp1, real-exp2, real-exp3*

yntax Rules

1. The optional *dev-id* must be preceded by a number sign (#) .
2. You specify the color index with *int-exp,* which must be greater than zero.
3. *Real-exp1* sets the value associated with red.
4. *Real-exp2* sets the value associated with green.
5. *Real-exp3* sets the value associated with blue.
6. Values for the intensities of the colors red, green, and blue must be greater than or equal to 0.0 and less than or equal to 1.0.

emarks

1. The optional *dev-id* identifies the device for which you want to set the color mix. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.

2. The default intensity values for color indices 1, 2, and 3 are as follow

Color Index	Color	Red Intensity	Green Intensity	Blue Intensity
0	Black	0.0	0.0	0.0
1	Green	0.0	1.0	0.0
2	Red	1.0	0.0	0.0
3	Blue	0.0	0.0	1.0

3. When you specify the intensities for red, green, and blue, you *mix* the colors for the index value. The color mix is dependent on the intensities you specify, not on the value of the index. For instance, tl following three statements set the indices 1, 2, and 3, to be the same color:

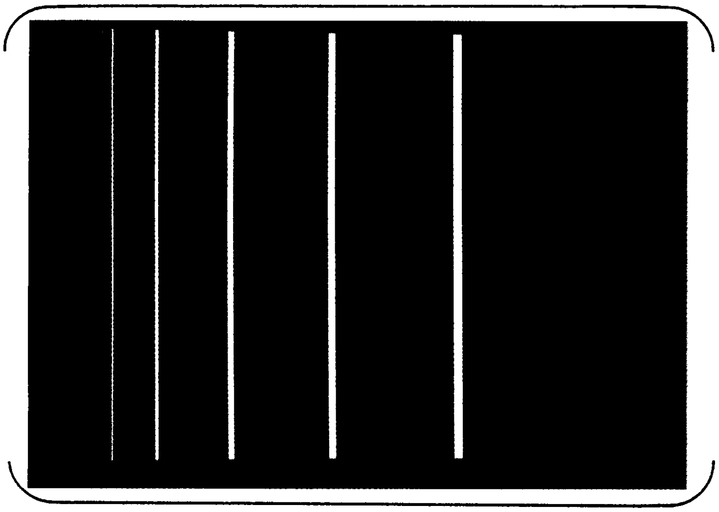
```
SET COLOR MIX ,INDEX 1 : 0.4281, 0.7119, 0.7119
SET COLOR MIX ,INDEX 2 : 0.4281, 0.7119, 0.7119
SET COLOR MIX ,INDEX 3 : 0.4281, 0.7119, 0.7119
```

4. If a device does not support the requested intensity, it will approximate the requested color.
5. Possible color intensities for red, green, and blue on VAXstations anc on VT125 and VT240 terminals are listed in Appendix B.

Example

```
DECLARE LONG growing,loop
SET WINDOW 0,120,0,120
FOR growing = 1 TO 10
  SET LINE SIZE growing
    SELECT growing
          CASE = 1,3,5,7,9
                 SET COLOR MIX , INDEX 1 : 1, 0.1400, 1
                 SET LINE COLOR 1
          CASE = 2,4,6,8,10
                 SET COLOR MIX , INDEX 2 : 1, 1, 0.4200
                 SET LINE COLOR 2
    END SELECT
  GRAPH LINES growing^2,120; growing^2,0
NEXT growing
END
```

Output



SET DEVICE VIEWPORT

The SET DEVICE VIEWPORT statement lets you change the boundaries of the device viewport.

Format

SET DEVICE VIEWPORT *[#dev-id :] real-exp1, real-exp2, real-exp3, real-exp4*

Syntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively.
2. The optional *dev-id* must be preceded by a number sign (#) and followed by a colon (:).

Remarks

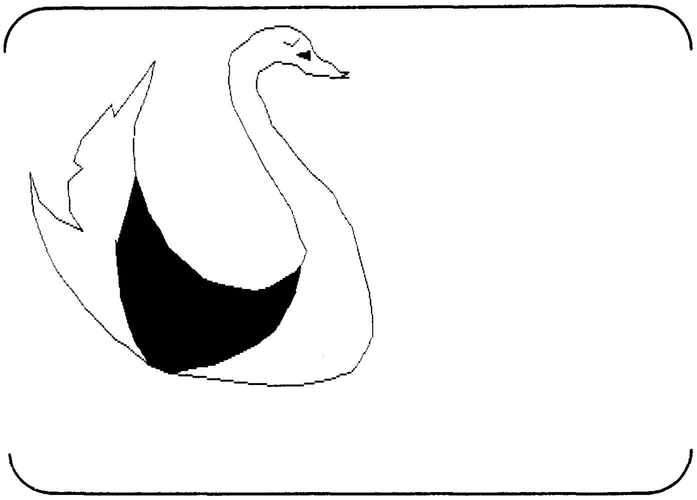
1. The device viewport is the portion of the device selected for a graphic display.
2. The optional *dev-id* identifies the device for which you want to set th viewport. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
3. The default device viewport is device dependent; however, it is usuall the largest square region the device can accommodate.
4. If the device viewport is not already clear, the SET DEVICE VIEWPORT statement clears the display surface.

5. When the device window and device viewport do not match in shape, VAX BASIC uses the largest possible device viewport that matches the shape of the device window. For instance, if you select a long thin device window and a square device viewport, VAX BASIC overrides your selection of a device viewport and uses the largest device viewport possible that is also long and thin, thus matching the device window as closely as possible.
6. Boundaries must be supplied as device coordinates. To determine the device coordinates for a particular device, use the ASK DEVICE SIZE statement.
7. You cannot change the clipping status at the device viewport boundaries. If an object is not within the device window, it is not displayed in the device viewport.
8. A SET DEVICE VIEWPORT statement is invalid in a picture definition.

xample

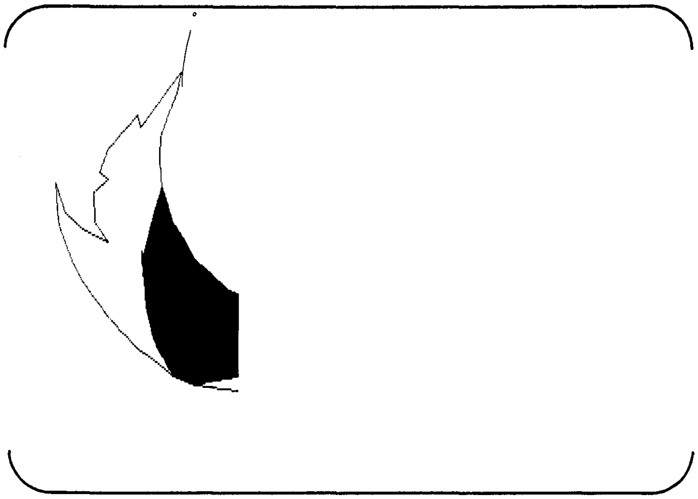
```
OPTION TYPE = EXPLICIT
DECLARE SINGLE left_1,right_1,bottom,top
EXTERNAL PICTURE swan
!+
!Set the world window - one swan fills entire NDC space
!-
SET WINDOW 0,100,0,100
Screen1:
 DRAW swan
 SLEEP 5%
 CLEAR
 ASK DEVICE VIEWPORT left_1,right_1,bottom,top
 !+
 !Set the device window to half of NDC space
 !-
 SET DEVICE WINDOW 0,0.5,0,1
 !+
 !Set the device viewport to the left half of the default
 !-
 SET DEVICE VIEWPORT left_1,right_1/2,bottom,top
Screen2:
 !+
 !After device transformation, only part of swan displayed
 !-
 DRAW swan
END
```

Output Screen 1:



ZK-4953-86

Output Screen 2:



ZK-4965-86

SET DEVICE WINDOW

The SET DEVICE WINDOW statement lets you change the boundaries of the device window.

ormat

SET DEVICE WINDOW *[#dev-id :] real-exp1, real-exp2, real-exp3, real-exp4*

yntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively. Values must be within the range 0.0 to 1.0.
2. The optional *dev-id* must be preceded by a number sign (#) and followed by a colon (:).

emarks

1. The device window is a portion of NDC space you select to be displayed on the device viewport of an active device.
2. The default device window is the entire NDC space with boundaries of 0,1,0,1.
3. The optional *dev-id* identifies the device for which you want to set the window. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
4. The boundaries you set must be within the range 0.0 to 1.0.
5. When the device window and device viewport do not match in shape, VAX BASIC uses the largest possible device viewport that matches the shape of the device window. For instance, if you select a long thin device window and a square device viewport, VAX BASIC overrides your selection of a device viewport and uses the largest

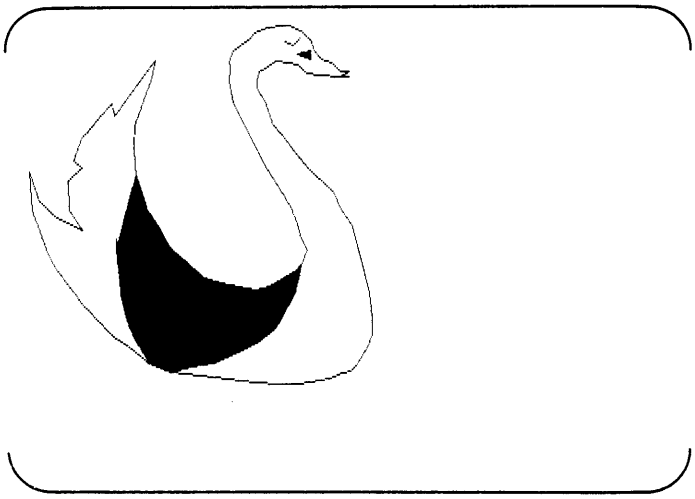
device viewport possible that is also long and thin, thus matching the device window as closely as possible.

6. Clipping of graphics objects outside the device window cannot be disabled. If an object is not within the device window, it is not displayed in the device viewport.
7. The SET DEVICE WINDOW statement clears the display surface if it is not already cleared.
8. A SET DEVICE WINDOW statement is invalid in a picture definition.

Example

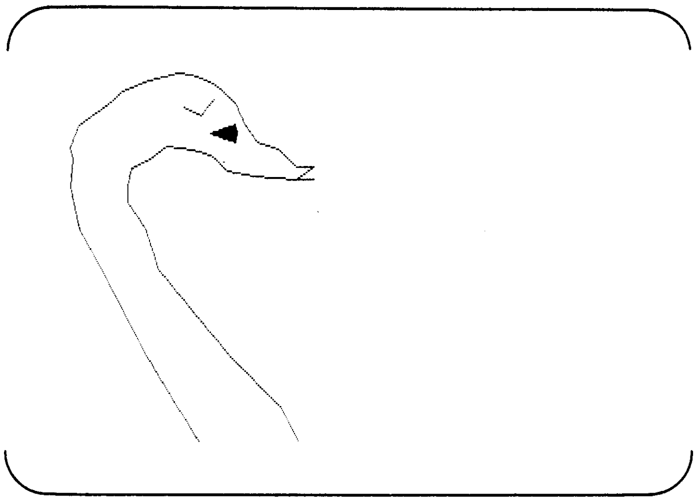
```
OPTION TYPE = EXPLICIT
DECLARE SINGLE left_1,right_1,bottom,top
EXTERNAL PICTURE swan
!+
!Set world window so that swan fills NDC space
!-
SET WINDOW , TRAN 1 : 0,100,0,100
Screen1:
 DRAW swan
 SLEEP 5%
 CLEAR
!+
!Select entire NDC space for world viewport
!-
SET VIEWPORT , TRAN 1 : 0,1,0,1
!+
!Select the top right square of NDC space as the device window
!but leave device viewport as default
!-
SET DEVICE WINDOW 0.5,1,0.5,1
Screen2:
 DRAW swan
END
```

Output Screen 1:



ZK-4953-86

Output Screen 2:



ZK-4959-86

SET INITIAL CHOICE

The SET INITIAL CHOICE statement allows you to specify items to be displayed in a menu as well as set up an initial selection for the user.

Format

```
                     ┌ #dev-id                                       ┐
                     │ , UNIT int-exp1                    ┌   ┐      │
SET INITIAL CHOICE   │ , COUNT int-exp2                   │ : │      │ int-exp3
                     │        { str-array              }  │   │      │
                     │ , LIST { (str-exp [ , str-exp ]...) } └   ┘      │
                     └                                               ┘
```

Syntax Rules

1. *Int-exp3* specifies the number of the menu item for the initial choice.
2. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
3. The optional LIST, COUNT, and UNIT clauses can be specified in any order.
4. If one or more optional clauses are included, one colon (:) is required before *int-exp3*.
5. *Str-array* must be a one-dimensional array containing string data types Virtual arrays are invalid.
6. *Int-exp2* cannot be greater than the number of elements in *str-array*.

emarks

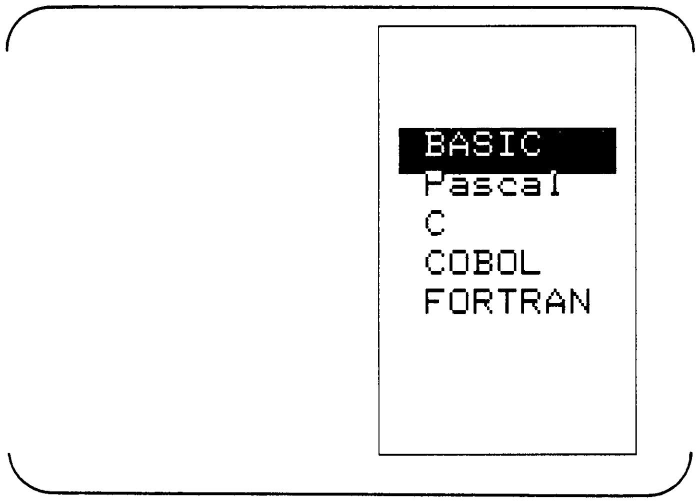
1. The optional *dev-id* identifies the device for which you want to set up a menu. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. The LIST clause specifies strings to be displayed in a menu. You can include a list of string expressions enclosed in parentheses, or an array containing the strings. When an array is included, specify only the array name; do not include parentheses to indicate the whole array. If no LIST clause is included, the default menu is displayed unless an alternative has been set with a previous SET INITIAL CHOICE statement.
3. The COUNT clause specifies the number of items in the string array to be displayed in the menu, starting with the lowest element in the array. If no string array is specified in the LIST clause, the COUNT clause is not valid and a compile-time error message is issued. The user cannot move the cursor to an item that is not displayed. If no COUNT clause is included, all of the menu items in the array elements are displayed.
4. The initial choice specified in a LOCATE CHOICE statement overrides the initial choice specified in a SET INITIAL CHOICE statement.
5. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
6. Subsequent requests for CHOICE input use the items specified until another SET INITIAL CHOICE statement sets an alternative menu.

Example

In the following example an array of 10 strings is supplied to the SET INITIAL CHOICE statement. The COUNT clause specifies that only the first five array elements (up to menu_items(4)) are to be displayed. The user cannot select a language that is not among the first five listed.

```
OPTION TYPE = EXPLICIT
DECLARE LONG language
DIM STRING menu_items(9)
menu_items(0) = "BASIC"
menu_items(1) = "Pascal"
menu_items(2) = "C"
menu_items(3) = "COBOL"
menu_items(4) = "FORTRAN"
menu_items(5) = "LISP"
menu_items(6) = "PL/1"
menu_items(7) = "SCAN"
menu_items(8) = "RPG/II"
menu_items(9) = "LOGO"
SET INITIAL CHOICE , COUNT 5              &
                   , LIST menu_items      &
                   : 1
LOCATE CHOICE language
   .
   .
   .
```

Output



ZK-4971-86

SET INITIAL MULTIPOINT

The SET INITIAL MULTIPOINT statement allows you to specify a set of initial points that a user can accept as the default input.

Format

```
                        [ #dev-id                    {   } ]
SET INITIAL MULTIPOINT  [ , UNIT int-exp1            { : } ]
                        [ , COUNT int-exp2           {   } ]
                        [ , USING TRAN int-exp3      {   } ]
                        x-array, y-array
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. The optional UNIT, COUNT, and USING TRAN clauses can be in an order.
3. *Int-exp3* must be an integer between 1 and 255.
4. If one or more optional clauses are included, one colon (:) is requirec before *x-array*.
5. For MULTIPOINT, specify only the array name; do not use parenthe- ses to indicate the whole array.
6. *X-array* and *y_array* must be one-dimensional arrays that contain onl integer or real data types. Virtual arrays are not valid.
7. *Int-exp2* cannot be greater than the number of elements in the coordi nate arrays.

emarks

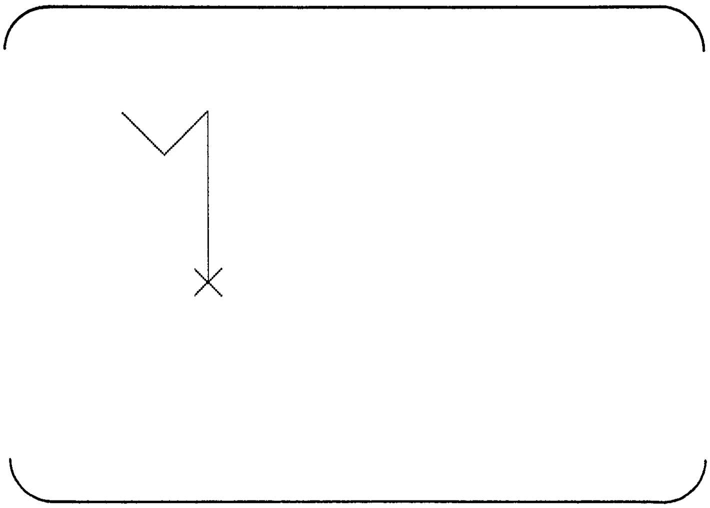
1. The optional *dev-id* identifies the device for which you want to set up the initial display. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. The COUNT clause allows you to display a selected number of points from the arrays starting with the first elements. If the COUNT clause is not specified, all of the points specified in the arrays are displayed.
3. Depending on the device, a user can delete the initial points with the DELETE key.
4. The USING TRAN clause allows you to specify an alternative transformation to be used to interpret the world coordinates of the initial points you specify. If no USING TRAN clause is included, the initial points are displayed using the currently established transformation.
5. Subsequent calls for MULTIPOINT input use the points specified until another SET INITIAL MULTIPOINT statement is executed or until the initial point is overridden by an AT clause in a MAT LOCATE POINTS or MAT GET POINTS statement.
6. If you do not include a SET INITIAL MULTIPOINT statement, VAX BASIC uses the initial point specified in a MAT LOCATE POINTS or MAT GET POINTS statement.
7. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
8. Execution of a MAT LOCATE POINTS or MAT GET POINTS statement causes the display of the initial points.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG CONSTANT max_points = 13
DECLARE LONG loop,how_many
DIM SINGLE maze_x(8), maze_y(8),                &
    SINGLE  input_xs(1 TO max_points), input_ys(1 TO max_points)
SET WINDOW , TRAN 1 : 0,100,0,100
DATA 0.1,0.8, 0.2,0.7, 0.3,0.8, 0.3,0.6,        &
     0.3,0.4, 0.2,0.5, 0.2,0.4, 0.1,0.4,        &
     0.1,0.2
READ maze_x(loop), maze_y(loop) FOR loop = 0 TO 8
SET INITIAL MULTIPOINT , COUNT 5                &
                       : maze_x, maze_y
MAT LOCATE POINTS , COUNT how_many : input_xs, input_ys
IF how_many > max_points
   THEN how_many = max_points
END IF
SET POINT COLOR 2
MAT GRAPH POINTS , COUNT how_many : input_xs, input_ys
END
```

Output

The output shows only the initial points presented to the user.


ZK-5522-86

ET INITIAL POINT

The SET INITIAL POINT statement allows you to specify an initial point that a user can enter as the default input.

ormat

```
                     [ #dev-id                  {   }  ]
SET INITIAL POINT    [ , UNIT int-exp1          { : }  ]  x-coord, y-coord
                     [ , USING TRAN int-exp2    {   }  ]
```

yntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. The optional UNIT and USING TRAN clauses can be in any order.
3. If one or more optional clauses are included, one colon (:) is required before *x-coord.*
4. *Int-exp2* must be an integer between 1 and 255.

emarks

1. The optional *dev-id* identifies the device for which you want to set the initial point. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly or opened by VAX BASIC as the default device.
2. The USING TRAN clause allows you to specify a transformation to interpret the world coordinates of the initial point you specify. If no USING TRAN clause is included, the initial point is displayed using the currently established transformation.

3. Subsequent calls for POINT input use the initial point specified until another initial point is set by a SET INITIAL POINT statement or a LOCATE/GET POINT statement.
4. The optional UNIT clause allows you to specify an alternative mean of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of the methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as th VAX GKS documentation.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE center_x, center_y, radius
DECLARE LONG which_tran
SET WINDOW , TRAN 1 : 0,100,0,100
SET WINDOW , TRAN 2 : 10,40,0,5
SET TRANSFORMATION 1
SET INITIAL POINT , USING TRAN 2              &
                  : 20,3.5
LOCATE POINT center_x, center_y , which_tran
LOCATE VALUE , RANGE 0.1 TO 0.4 : radius
   .
   .
   .
END
```

ET INITIAL STRING

The SET INITIAL STRING statement allows you to specify an initial string expression that is presented to a user when string input is requested. A user can enter the initial string or enter another string.

ormat

$$\textbf{SET INITIAL STRING} \left[\begin{matrix} \#\textit{dev-id} \\ \textbf{, UNIT } \textit{int-exp} \end{matrix} \left\{ : \right\} \right] \textit{str-exp}$$

yntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. If one or more optional clauses are included, one colon (:) is required before *str-exp*.

emarks

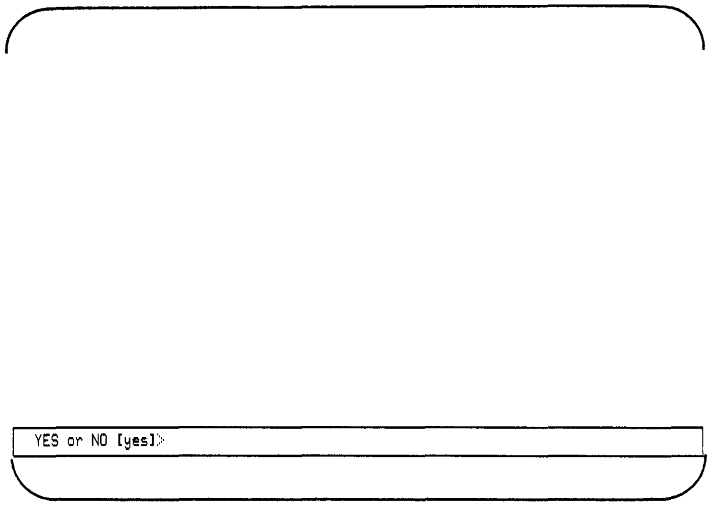
1. The optional *dev-id* identifies the device for which you want to set the initial string. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly, or opened by VAX BASIC as the default device.
2. Subsequent requests for STRING input use the item specified until another SET INITIAL STRING statement or LOCATE STRING statement specifies an alternative.

3. VAX BASIC appends the user-supplied string to the initial string you supply in a SET INITIAL STRING or a LOCATE STRING statement. If the user accepts the initial string with a null response, *str-var* contains only the initial string. Depending on the device, a user may be able to delete the initial string by pressing CTRL/U or using the DELETE key.
4. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse or with the keyboard arrow keys. Each of thes methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as th VAX GKS documentation.

Example

```
PROGRAM select_an_item
  DECLARE STRING CONSTANT init_string =              &
          "YES or NO [yes]>"
  DECLARE STRING user_string
  EXTERNAL PICTURE Main_menu, Input_menu
  OPEN "my_term" FOR GRAPHICS AS DEVICE #1
   .
   .
   .
  SET INITIAL STRING #1 : init_string
  LOCATE STRING #1 : user_string
  IF init_string = user_string
     THEN
     DRAW Main_menu
     ELSE
     DRAW Input_menu
  END IF
   .
   .
   .
```

Output



```
YES or NO [yes]:>
```

ZK-5523-86

SET INITIAL VALUE

The SET INITIAL VALUE statement allows you to specify an initial floating-point number within a specified range. A user can enter this value or an alternative.

Format

```
                    [ #dev-id                               {   }  ]
SET INITIAL VALUE   [ , UNIT int-exp                        { : }  ]  real-exp3
                    [ , RANGE real-exp1 TO real-exp2        {   }  ]
```

Syntax Rules

1. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
2. The optional UNIT and RANGE clauses can be in any order.
3. *Real-exp1* must be less than *real-exp2* in the optional RANGE clause.
4. If one or more optional clauses are included, one colon (:) is required before *real-exp3*.
5. You specify the initial value in *real-exp3*.

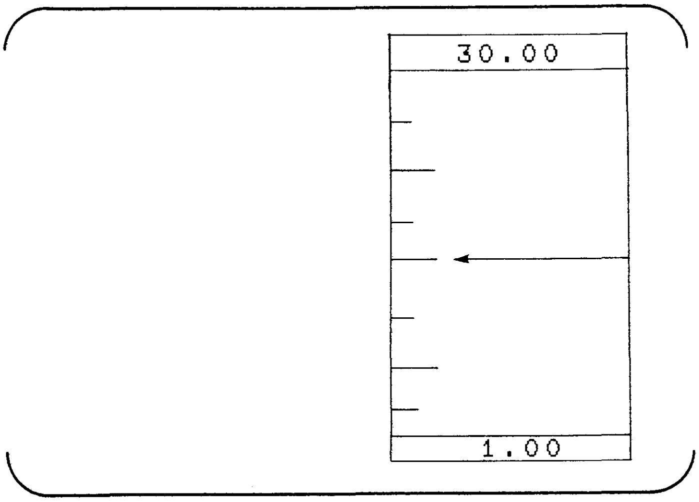
Remarks

1. The optional *dev-id* identifies the device for which you want to set an initial value. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly or opened by VAX BASIC as the default device.
2. Subsequent requests for value input with the LOCATE VALUE statement use the initial value and range specified until another INITIAL VALUE or LOCATE VALUE statement specifies another value.
3. The RANGE clause specifies the lowest and highest acceptable input. The user-supplied value and the initial value must be within this range. VAX BASIC signals the run-time error ILLINIVAL, "Illegal initial value" (ERR = 284), when the initial value is not within the specified range.
4. When the LOCATE VALUE statement is executed, the user has the option of entering the initial value specified in this statement or another value as input.
5. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL SUB Graphics_display(SINGLE)
DECLARE SINGLE which_val
OPEN "VT124" FOR GRAPHICS AS DEVICE #1
    .
    .
    .
Input_rtn:
  SET INITIAL VALUE #1                            &
                    , RANGE  1 TO 30              &
                    : 15
LOCATE VALUE #1 : which_val
IF which_val =< 15
   THEN GOTO Large_size
   ELSE CALL Graphics_display(which_val)
END IF
    .
    .
    .
```

Output



ZK-5230-86

ET INPUT PRIORITY

The SET INPUT PRIORITY statement allows you to modify the transformation priority list.

ormat

$$\textbf{SET INPUT PRIORITY} \quad \begin{Bmatrix} \textit{int-var1} \\ \textit{int-const1} \end{Bmatrix} \begin{Bmatrix} < \\ > \end{Bmatrix} \begin{Bmatrix} \textit{int-var2} \\ \textit{int-const2} \end{Bmatrix}$$

yntax Rules

The constant or variable values must be between 1 and 255. Expressions are not allowed.

emarks

1. When points are input from a device, VAX BASIC maps these points to world coordinates according to the specifications for the transformation that the point or points fall in.

   When more than one transformation is defined, world viewports can overlap. The viewport with the highest priority that contains all of the points is used to map the points back to world coordinates. For more information, see Chapter 7.
2. Initially, the transformation with the highest priority is transformation 1. Transformation 1 is always defined and initially includes the entire NDC space.
3. When other transformations are defined, the highest input priority is the transformation that has been most recently established with a SET INPUT PRIORITY, SET WINDOW, SET VIEWPORT, or SET TRANSFORMATION statement.

4. The less-than sign (<) sets the first value to be a lower priority than the second. The greater-than sign (>) sets the first value to be a higher priority than the second.
5. Changing the input priority does not change the transformation currently in effect for displaying output.
6. When transformation 1 has been redefined with a viewport other than 0,1,0,1, a point can be input beyond the limits of any defined viewport. When this happens, VAX BASIC signals an error.
7. A SET INPUT PRIORITY statement is invalid within a picture definition.

Example

```
DECLARE LONG how_many,counter
DIM LONG xforms(10)
SET WINDOW ,TRAN 1 : 1000,2000,0,100
SET VIEWPORT ,TRAN 1 : 0,1,0,1
SET WINDOW ,TRAN 2 : 0,100,0,100
SET VIEWPORT ,TRAN 2 : 0,1,0,0,0.5
SET WINDOW , TRAN 3 : -1,1,-1,1
SET VIEWPORT , TRAN 3 : 0.5,1,0.5,1
!+
!Show tran priority list
!-
ASK TRANSFORMATION LIST , COUNT how_many : xforms
PRINT xforms(counter) FOR counter = 0 TO (how_many - 1)
!+
!Set highest priority to TRAN 2 for input
!-
SET INPUT PRIORITY 2 > 3
!+
!Show new priority list
!-
ASK TRANSFORMATION LIST , COUNT how_many : xforms
PRINT xforms(counter) FOR counter = 0 TO (how_many - 1)
END
```

Output

```
 3
 2
 1

 2
 3
 1
```

;ET LINE COLOR

The SET LINE COLOR statement allows you to specify a color for lines.

ormat

SET LINE COLOR *int-exp*

yntax Rules

Int-exp must be a valid color value for all active devices, and must be greater than zero.

emarks

1. Changing the line color affects the display of lines on all active devices. You cannot specify a particular device with this statement.
2. Changing color values may affect the shading on some black and white devices.
3. The actual colors displayed are device dependent. If you set the color to an index value that is undefined for a particular device, results are unpredictable.
4. On VT125 and VT240 terminals with the color option, the default values for *int-exp* represent the following colors:

Index	Color
0	Black
1	Green
2	Red
3	Blue

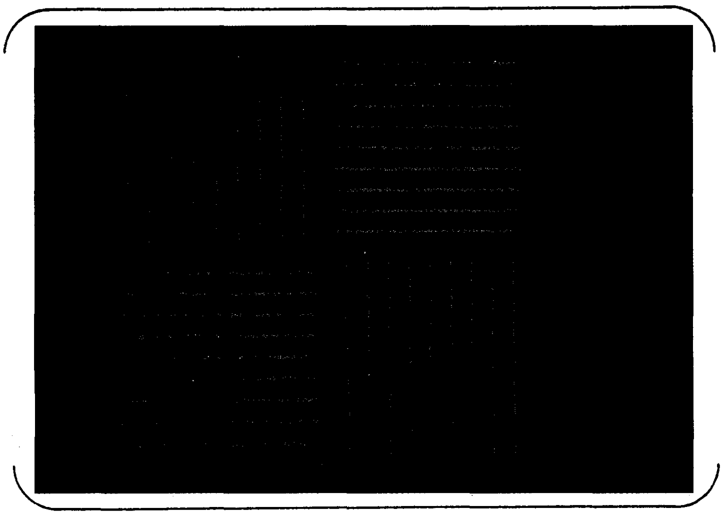
5. See also the SET COLOR MIX statement for related information.

SET LINE COLOR

Example

```
PICTURE line_display(LONG color_var)
  DECLARE SINGLE loop
  SET LINE COLOR color_var
  SET LINE SIZE 3
  PLOT LINES 0.05,loop; 0.95,loop FOR loop = 0.1 TO 1 STEP 0.1
END PICTURE

PROGRAM Show_lines
  OPTION ANGLE = DEGREES
  EXTERNAL PICTURE line_display(LONG)
  SET WINDOW -1,1,-1,1
  DRAW line_display(1)
  DRAW line_display(2) WITH ROTATE(90)
  DRAW line_display(1) WITH ROTATE(180)
  DRAW line_display(2) WITH ROTATE(270)
END PROGRAM
```

Output



SET LINE SIZE

The SET LINE SIZE statement allows you to select the current width of line.

Format

SET LINE SIZE *num-exp*

Syntax Rules

Num-exp must be greater than zero.

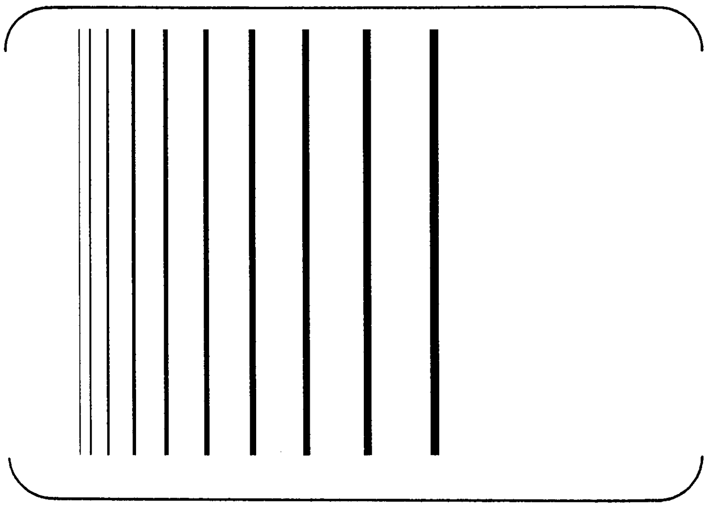
Remarks

1. *Num-exp* is the scale factor that is multiplied by the smallest possible width of a line.
2. Subsequent line output is drawn with the size you set until another SET LINE SIZE statement is executed.
3. The default size for the line width is 1. This is the minimum size a device can produce.
4. The range and number of possible line widths are device dependent. To determine the maximum size for a particular device, use the ASK MAX LINE SIZE statement. If you set the size to a value that is undefined for the device, results are unpredictable.

Example

```
DECLARE LONG growing,loop
SET WINDOW 0,120,0,120
FOR growing = 1 TO 10
  SET LINE SIZE growing
  GRAPH LINES growing^2,120; growing^2,0
NEXT growing
END
```

Output



ZK-5521-86

SET LINE STYLE

The SET LINE STYLE statement lets you specify the style with which lines are drawn.

Format

SET LINE STYLE *int-exp*

Syntax Rules

None.

Remarks

1. The numeric values for line styles represent the following styles:

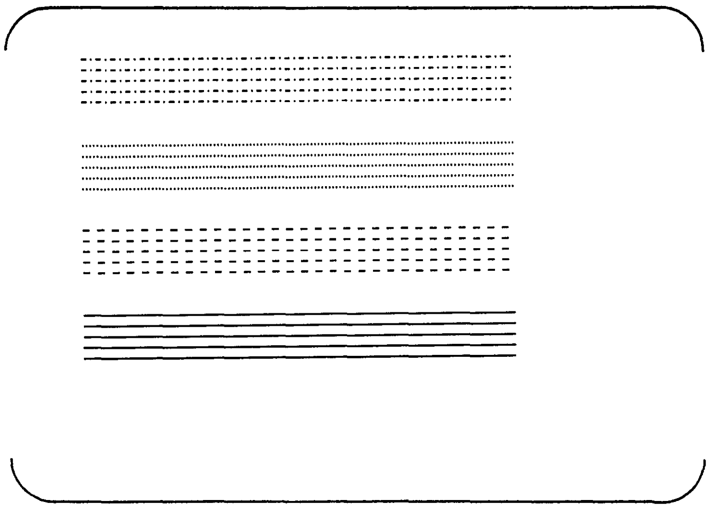
Value	Line Style
1	Solid (default)
2	Dashed
3	Dotted
4	Dashed-dotted

   Line styles greater than 4 are device dependent. If you set the style to a value that is not supported by the device, results are unpredictable.
2. The default line style is 1, solid.
3. Subsequent line output is displayed with the style you set until another SET LINE STYLE statement is executed.

Example

```
PROGRAM Demo
 OPTION TYPE = EXPLICIT
 EXTERNAL PICTURE line_demo(LONG)
 DECLARE LONG line_style
 DRAW line_demo(Line_style) FOR line_style = 1 TO 4
END PROGRAM

PICTURE line_demo(LONG line_style)
  SET LINE STYLE line_style
  GRAPH LINES 0,line_style/5;  1,line_style/5
  GRAPH LINES 0,(line_style/5) + 0.025;  1,(line_style/5) + 0.025
  GRAPH LINES 0,(line_style/5) + 0.055  1,(line_style/5) + 0.05
  GRAPH LINES 0,(line_style/5) + 0.075;  1,(line_style/5) + 0.075
  GRAPH LINES 0,(line_style/5) + 0.1;  1,(line_style/5) + 0.1
END PICTURE
```

Output



SET POINT COLOR

The SET POINT COLOR statement allows you to specify the color for points.

Format

SET POINT COLOR *int-exp*

Syntax Rules

Int-exp must be a valid color value for all devices in use, and must be greater than zero.

Remarks

1. Changing color attributes affects the display of graphics objects on all active devices. You cannot specify a particular device with this statement.
2. Changing color values may affect the shading on some black and white devices.
3. The actual colors displayed are device dependent. If you set the color to an index value that is undefined for a particular device, results are unpredictable.

4. On VT125 and VT240 terminals with the color option, the default values for *int-exp* represent the following colors:

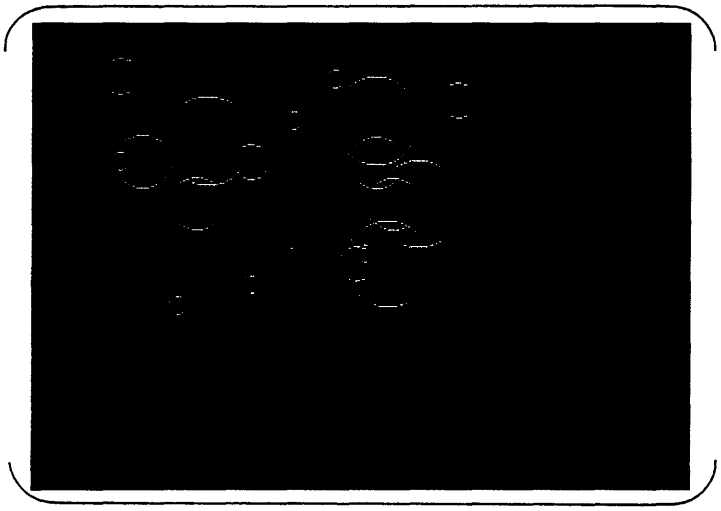
Index	Color
0	Black
1	Green
2	Red
3	Blue

5. See also the SET COLOR MIX statement for related information.

Example

```
PROGRAM bubbles
  OPTION TYPE = EXPLICIT
  EXTERNAL PICTURE markers(LONG,LONG)
  DECLARE LONG CONSTANT blue = 3, pink = 2
  DRAW markers(blue,4)
  SLEEP 4%
  CLEAR
  SET COLOR MIX , INDEX pink : 0.6258, 0.2142, 0.2142
  DRAW markers(pink,4)
END PROGRAM

PICTURE markers(LONG color_var, LONG style_var)
  SET POINT COLOR color_var
  SET POINT STYLE style_var
  SET POINT SIZE 2
  PLOT POINTS 0.1,0.9; 0.4,0.7; 0.9,0.85; 0.65,0.45
  SET POINT SIZE 1
  PLOT POINTS 0.5,0.8; 0.6,0.9; 0.1,0.7
  SET POINT SIZE 5
  PLOT POINTS 0.8,0.6; 0.3,0.75; 0.7,0.8; 0.725,0.45
  SET POINT SIZE 3
  PLOT POINTS 0.15,0.7; 0.74,0.6; 0.7,0.7; 0.275,0.6
  SET POINT SIZE 1
  PLOT POINTS 0.23,0.35; 0.4,0.4; .67,0.475
END PICTURE
```

Output



SET POINT SIZE

The SET POINT SIZE statements allow you to select the current size of a point.

Format

SET POINT SIZE *num-exp*

Syntax Rules

Num-exp must be greater than zero.

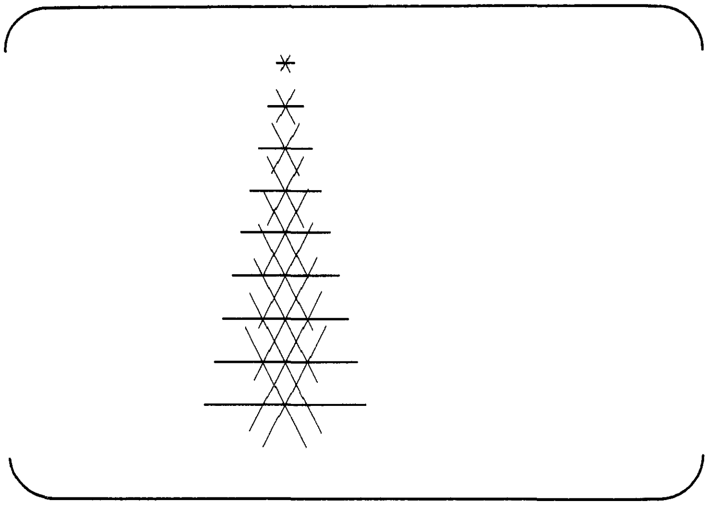
Remarks

1. *Num-exp* is the scale factor that is multiplied by the smallest possible size of a point.
2. Subsequent point output is drawn with the size you set until another SET POINT SIZE statement is executed.
3. The default size for points is 1. This is the minimum size a device can produce.
4. Regardless of what size you set for a point size, points set to the dot marker are always drawn in the smallest possible size.
5. The range and number of possible point sizes are device dependent. To determine the maximum size for a particular device, use the ASK MAX POINT SIZE statements. If you set the size to a value that is undefined for the device, results are unpredictable.

Example

```
DECLARE LONG loop
SET WINDOW 0,10,0,10
FOR loop = 1 TO 9
  SET POINT SIZE loop
  GRAPH POINTS 5,(10 - loop)
NEXT loop
END
```

Output



ZK-5530-86

SET POINT STYLE

The SET POINT STYLE statement lets you specify your choice of style for the marker for a point.

Format

SET POINT STYLE *int-exp*

Syntax Rules

None.

Remarks

1. The numeric values for point styles represent the following markers:

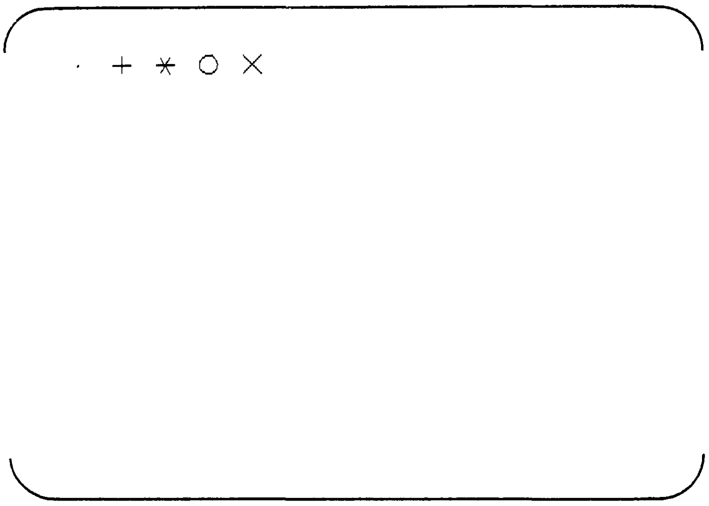
Value	Point Style
1	Dot
2	Plus sign
3	Asterisk (default)
4	Circle
5	Diagonal cross

   Values greater than 5 are device dependent. If you set the style to a value that is not supported by the device, results are unpredictable.
2. The default point style is 3, an asterisk.
3. Subsequent points are displayed with the style you set until another SET POINT STYLE statement is executed.

Example

```
PROGRAM Demo
OPTION TYPE = EXPLICIT
DECLARE LONG point_style
FOR point_style = 1 TO 5
  SET POINT STYLE point_style
  GRAPH POINTS point_style/10,0.8
NEXT point_style
END PROGRAM
```

Output



SET STRING ECHO AREA

The SET STRING ECHO AREA statement lets you specify the boundaries of the STRING echo area.

Format

$$\textbf{SET STRING ECHO AREA} \left[\begin{matrix} \#\textit{dev-id} \\ \textbf{, UNIT}\ \textit{int-exp} \end{matrix} \left\{ : \right\} \right]$$

real-exp1, real-exp2, real-exp3, real-exp4

Syntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively.
2. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
3. If one or more optional clauses are included, one colon (:) is required before *real-exp1.*

Remarks

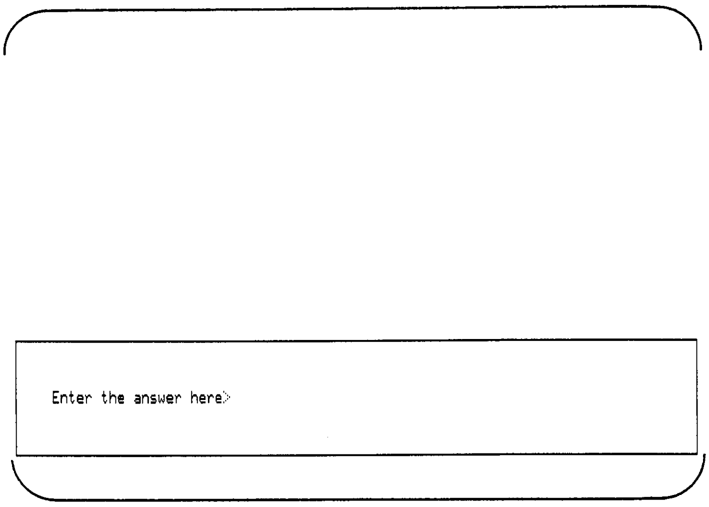
1. The echo area is the portion of your screen where the prompt appears and where input can be supplied by a user.
2. The optional *dev-id* identifies the device for which you want to set the echo area. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly or opened by VAX BASIC as the default device.

3. Boundaries must be specified in device coordinates. To determine the possible values on a particular device, use the ASK DEVICE SIZE statement. To determine the current boundaries, use the ASK STRING ECHO AREA statement. Appendix B lists the boundaries for VAXstations and VT125 and VT240 terminals.
4. The default echo area for STRING input is device dependent. The default area is in effect until a SET STRING ECHO AREA statement is executed.
5. Subsequent requests for each input type use the echo boundaries you supply until another SET STRING ECHO AREA statement is executed.
6. Be aware that the text height cannot be reduced lower than the smallest text height available for the device. Therefore, you should ensure that the STRING echo area is large enough for the text characters you specify.
7. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with a mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.

Example

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE left_1,right_1,bottom,top
DECLARE STRING user_string
OPEN "office-term" FOR GRAPHICS AS DEVICE #1
   .
   .
   .
ASK STRING ECHO AREA #1 : left_1,right_1,bottom,top
!+
!Set the new echo area
!-
SET STRING ECHO AREA #1 : left_1             &
                      ,right_1               &
                      ,bottom                &
                      ,top * 4
SET INITIAL STRING #1 : "Enter the answer here>"
LOCATE STRING #1 : user_string
   .
   .
   .
```

Output



ZK-5534-86

SET TEXT ANGLE

The SET TEXT ANGLE statement lets you change the angle with which a text string is rotated.

ormat

SET TEXT ANGLE *real-exp*

yntax Rules

Real-exp must be an angle in radians or degrees, as specified in an OPTION ANGLE statement; the default is radians.

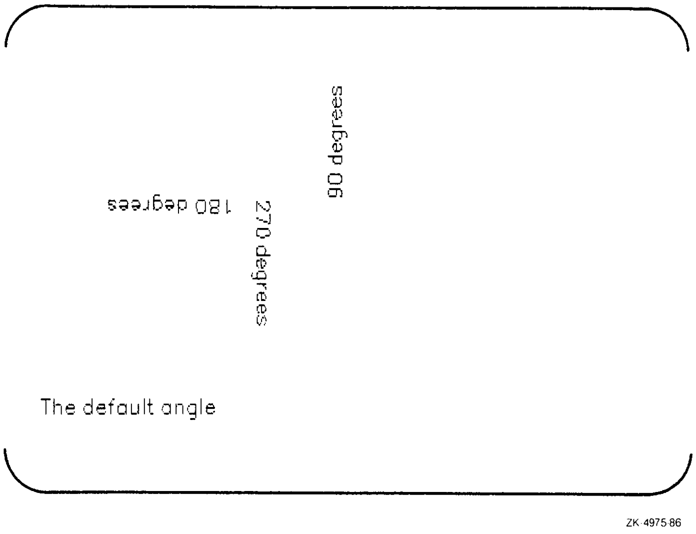
emarks

1. The default text angle is zero. Characters drawn with the default text angle are drawn horizontally across the screen.
2. Text is rotated about the starting point indicated in the GRAPH TEXT statement.
3. When you set the angle to a positive value, the text is rotated in a counterclockwise direction; supplying negative values rotates the text in a clockwise direction.
4. Angles that are integer multiples of 180° display text horizontally.
5. Angles that are integer multiples of 90° display text vertically.

Example

```
OPTION ANGLE = DEGREES
OPTION TYPE = EXPLICIT
SET TEXT FONT -3, "STROKE"
SET TEXT HEIGHT 0.05
SET TEXT ANGLE 90
GRAPH TEXT AT 0.7,0.55 : "90 degrees"
SET TEXT ANGLE 180
GRAPH TEXT AT 0.45,0.55 : "180 degrees"
SET TEXT ANGLE 270
GRAPH TEXT AT 0.5,0.55 : "270 degrees"
SET TEXT ANGLE 0
GRAPH TEXT AT 0.0,0.05 : "The default angle"
```

Output



;ET TEXT COLOR

The SET TEXT COLOR statement allows you to specify the color of text characters.

ormat

SET TEXT COLOR *int-exp*

yntax Rules

Int-exp must be a valid color value for all devices in use, and must be greater than zero.

emarks

1. Changing the text color affects the display of text on all active devices. You cannot specify a particular device with this statement.
2. Changing color values may affect the shading on some black and white devices.
3. The actual colors displayed are device dependent. If you set the color to an index value that is undefined for a particular device, results are unpredictable.
4. On VT125 and VT240 terminals with the color option, the default values for *int-exp* represent the following colors:

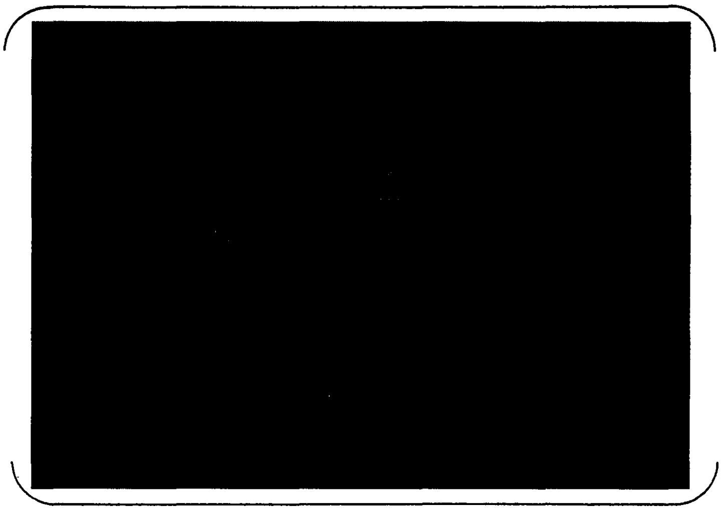
Index	Color
0	Black
1	Green
2	Red
3	Blue

5. See also the SET COLOR MIX statement for related information.

Example

```
PROGRAM colorful
  DECLARE SINGLE concat_x,concat_y
  SET TEXT HEIGHT 0.2
  SET TEXT COLOR 2
  GRAPH TEXT AT 0.0,0.5 : "C"
  ASK TEXT POINT , "C" AT 0.0,0.5 : concat_x, concat_y
  SET TEXT COLOR 3
  GRAPH TEXT AT concat_x, concat_y : "o"
  ASK TEXT POINT , "o" AT concat_x, concat_y: concat_x, concat_y
  SET TEXT COLOR 1
  GRAPH TEXT AT concat_x, concat_y : "l"
  ASK TEXT POINT , "l" AT concat_x, concat_y : concat_x, concat_y
  SET TEXT COLOR 2
  GRAPH TEXT AT concat_x, concat_y : "o"
  ASK TEXT POINT , "o" AT concat_x, concat_y : concat_x, concat_y
  SET TEXT COLOR 3
  GRAPH TEXT AT concat_x, concat_y : "r"
  ASK TEXT POINT , "r" AT concat_x, concat_y :concat_x, concat_y
  SET TEXT COLOR 1
  GRAPH TEXT AT concat_x, concat_y : "f"
  ASK TEXT POINT , "f" AT concat_x, concat_y : concat_x, concat_y
  SET TEXT COLOR 2
  GRAPH TEXT AT concat_x, concat_y : "u"
  ASK TEXT POINT , "u" AT concat_x, concat_y : concat_x, concat_y
  SET TEXT COLOR 3
  GRAPH TEXT AT concat_x, concat_y : "l"
END PROGRAM
```

Output



SET TEXT EXPAND

The SET TEXT EXPAND statement lets you change the ratio of character width to height from the ratio defined in the original font design.

Format

SET TEXT EXPAND *real-exp*

Syntax Rules

Real-exp must be greater than zero.

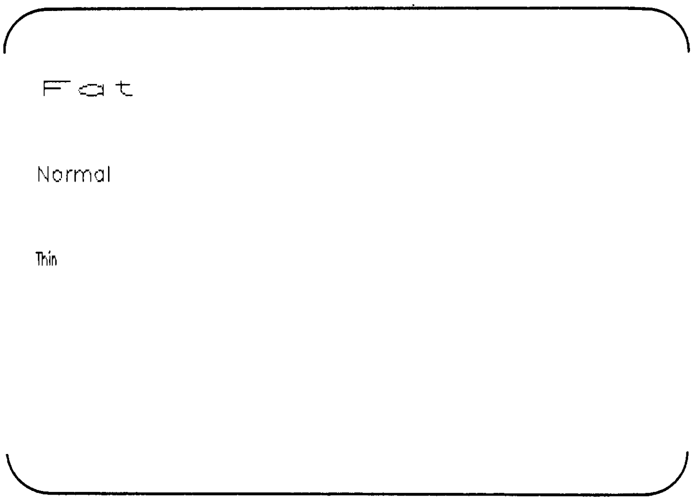
Remarks

1. The default width to height ratio is 1.0, which specifies the expansior ratio defined in the font design.
2. The value you assign to *real-exp* is the expansion factor used in subsequent GRAPH TEXT statements and stays in effect until anothei SET TEXT EXPAND statement is executed.

Example

```
SET TEXT FONT -3, "STROKE"
SET TEXT HEIGHT 0.05
SET TEXT EXPAND 3
GRAPH TEXT AT 0,0.8 : "Fat"
SET TEXT EXPAND 1
GRAPH TEXT AT 0,0.6 : "Normal"
SET TEXT EXPAND 0.5
GRAPH TEXT AT 0,0.4 : "Thin"
```

Output



ZK-4981-86

SET TEXT FONT

The SET TEXT FONT statement lets you select a font and specify the le of precision with which the characters are drawn.

Format

SET TEXT FONT *int-exp [, str-exp]*

Syntax Rules

1. *Int-exp* specifies the font number.
2. The optional *str-exp* specifies the level of precision with which the font characters are drawn.

Remarks

1. The default font is font -1, which is displayed with STROKE precis
2. Hardware fonts are device dependent. When no hardware fonts are available, software fonts are used. Software fonts are device independent and provide a graphical representation of the defined characters. Depending on the device, hardware fonts are usually displayed faster than software fonts.
3. If the precision string is not included, the specified software font is displayed with STROKE precision by default.
4. Hardware fonts can be displayed with either STRING or CHAR precision. If a hardware font is requested with STROKE precision, software font that is closest in number to the font you specify will displayed.
5. Software fonts can be drawn only with STROKE precision which provides an accurate representation of the text characters as specifi in the font design. Software fonts can be displayed with only STRC precision. If you specify an alternative precision string, a hardwar font will be displayed.

6. The number of hardware and software fonts is device dependent. If you specify that a font be displayed with a precision that is not available on a particular device, an alternative font will be displayed with the precision you specify. VT125 and VT240 fonts are displayed throughout this manual. VAXstation fonts are displayed in Appendix B.
7. **STRING Precision**

   STRING precision is available only for hardware fonts. STRING precision uses the device-specific character style defined in the font design. The starting position of the string is guaranteed to start at the same point as is defined in the font design. However, attributes such as the text path, alignment, angle, and spacing are not guaranteed to match the specifications in the font design.

   When clipping is enabled, STRING precision clips text on a string basis at the world viewport boundary, This means that no text string starts beyond the world viewport boundary.
8. **CHAR Precision**

   CHAR precision is available only for hardware fonts. This precision value uses the device-specific character style defined in the font design, as STRING precision does. However, this precision guarantees the text height, character expansion factor, spacing, angle, and alignment as much as is possible for a given device.

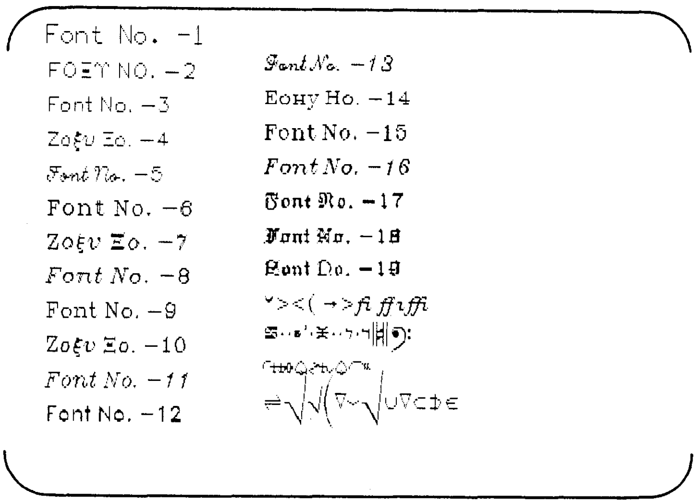
   When clipping is enabled, CHAR precision clips text at least on a character-by-character basis at the world viewport. This means that no character extends beyond the world viewport boundary. Depending on the capabilities of the device, character clipping is either partial or complete.
9. **STROKE Precision**

   STROKE precision is available only for software fonts. STROKE precision displays characters with the most accuracy. In addition, clipping is more precise with STROKE precision. When clipping is enabled, STROKE precision clips text precisely at the world viewport boundary. This means that if only half a character falls inside the world viewport boundary, only this half is displayed on the screen.

Example

```
OPTION TYPE = EXPLICIT
DECLARE LONG variety
SET WINDOW 0,100,0,100
SET TEXT HEIGHT 3
FOR variety = -1 TO -23 STEP -1
  SET TEXT FONT variety , "STROKE"
  IF variety > -13
  THEN
  GRAPH TEXT AT 0,100-(ABS(variety) * 8) : "Font No. " + STR$(variety)
  ELSE
  GRAPH TEXT AT 50,100-((ABS(variety) * 8) - 90) : "Font No. " + STR$(variet
  END IF
NEXT variety
END
```

Output



ZK 4878 86

:T TEXT HEIGHT

The SET TEXT HEIGHT statement lets you change the height of the characters in a font.

'mat

ET TEXT HEIGHT *real-exp*

ıtax Rules

Real-exp must be greater than zero.

marks

1. Height refers to the height of the uppercase letters in world coordinates. Lowercase characters are sized proportionally.
2. The value of *real-exp* must be greater than zero.
3. At the start of program execution, the text height is 0.035.
4. When you change the boundaries of the world window, you may need to adjust the text height to a value that is suitable for the new window size. To regain a text height equivalent to the default height after setting the window, use the following formula:

   ```
   new_height = (max_height - min_height) * 0.035
   ```

   For example:

   ```
   SET WINDOW 0,100,1000,5000
   new_height = (5000 - 1000) * 0.035
   SET TEXT HEIGHT new_height
   ```

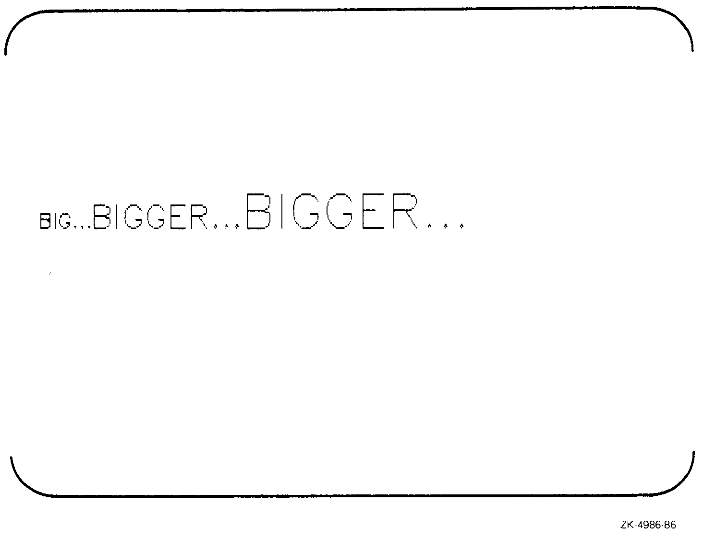
5. For CHAR and STRING precision, the text height is set to the nearest possible value to your request. For STROKE precision, text size is matched accurately.

6. When you invoke a picture with transformation functions in the DRAW statement, the text height and the text starting point are no automatically adjusted. You account for these transformations by using the TRANSFORM function, as shown in the example for the TRANSFORM function in this chapter.

Example

```
SET WINDOW 0,100,0,100
SET TEXT FONT -3, "STROKE"
SET TEXT HEIGHT 4
GRAPH TEXT AT 0,50 : "BIG..."
SET TEXT HEIGHT 7
GRAPH TEXT AT 10,50 : "BIGGER..."
SET TEXT HEIGHT 10
GRAPH TEXT AT 40,50 :"BIGGER..."
```

Output



T TEXT JUSTIFY

The SET TEXT JUSTIFY statement allows you to change the values of the current horizontal and vertical components of text justification.

mat

ET TEXT JUSTIFY *str-exp1, str-exp2*

ıtax Rules

1. You specify the value for the horizontal component in *str-exp1* and the value for the vertical component in *str-exp2*.
2. Possible values for the horizontal component in *str-exp1* are as follows:

Value	Effect on Horizontal Component
LEFT	Corresponds to the left side of the text box passing through the text position
CENTER	Corresponds to the text position lying midway between the left and right sides of the text box
RIGHT	Corresponds to the right side of the text box passing through the text position
NORMAL	Depends on the text path—see remark #3

3. Possible values for the vertical component in *str-exp2* are as follow

Value	Effect on Vertical Component
TOP	The top of the text box passes through the text position.
CAP	The text position passes through the capline of the whol string.
HALF	The text position passes through the half-line of the whc string.
BASE	The text position lies on the baseline of the whole string
BOTTOM	The bottom of the text box passes through the text.
NORMAL	See remark #3.

Remarks

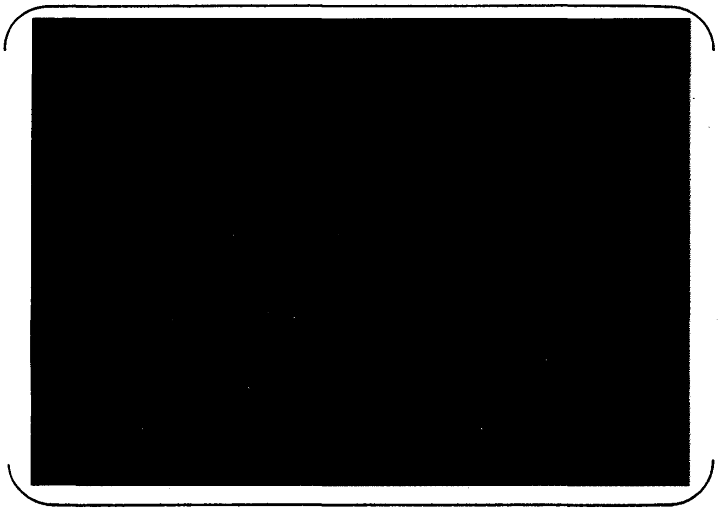
1. When a SET TEXT JUSTIFY statement is executed, the two string variables are evaluated to establish an imaginary box surrounding subsequent text output. The box is relative to the starting point specified in a GRAPH TEXT statement. Text justification has two components: horizontal and vertical.
2. The initial value for both the horizontal and the vertical componer is NORMAL.
3. NORMAL can be specified for both the horizontal and vertical con ponents. NORMAL provides a natural justification for each text pa For each of the text paths the horizontal and vertical components a as follows:

Text Path	Normal Horizontal	Normal Vertical
RIGHT	LEFT	BASE
LEFT	RIGHT	BASE
UP	CENTER	BASE
DOWN	CENTER	TOP

ample

```
SET TEXT FONT -3, "STROKE"
SET TEXT HEIGHT 0.04
SET TEXT COLOR 2
SET LINE COLOR 3
SET TEXT JUSTIFY "NORMAL" , "TOP"
GRAPH TEXT AT 0.4,0.9 : "Normal/top"
GRAPH LINES 0.4,0.9; 0.7,0.9

SET TEXT JUSTIFY "NORMAL" , "BOTTOM"
GRAPH TEXT AT 0.4,0.7 : "Normal/bottom"
GRAPH LINES 0.4,0.7; 0.7,0.7

SET TEXT JUSTIFY "LEFT" , "NORMAL"
GRAPH TEXT AT 0.4,0.5 : "Left/normal"
GRAPH LINES 0.4,0.5; 0.7,0.5

SET TEXT JUSTIFY "CENTER" , "NORMAL"
GRAPH TEXT AT 0.4,0.3 : "Center/normal"
GRAPH LINES 0.4,0.3; 0.7,0.3
SET TEXT JUSTIFY "RIGHT" , "NORMAL"
GRAPH TEXT AT 0.4,0.1 : "Right/normal"
GRAPH LINES 0.4,0.1; 0.7,0.1
```

Output



SET TEXT PATH

The SET TEXT PATH statement lets you specify the direction in which characters are drawn.

Format

SET TEXT PATH *str-exp*

Syntax Rules

Str-exp can be one of the following values (in lower- or uppercase):

- RIGHT
- LEFT
- UP
- DOWN

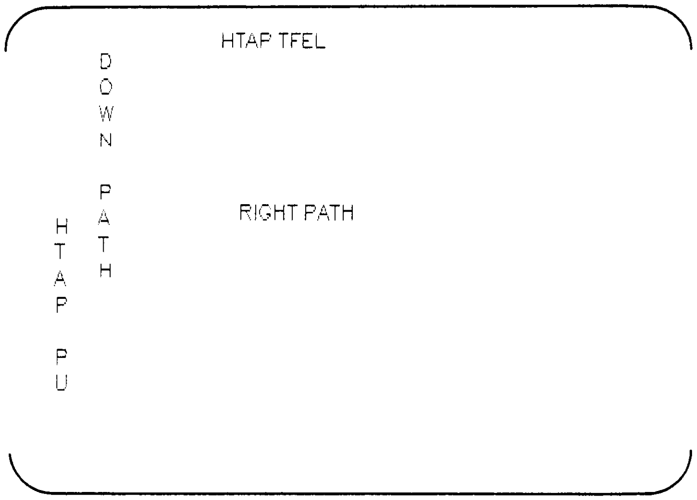
Remarks

1. Path refers to the direction in which the text is written relative to th text angle.
2. The default value for *str-exp* is RIGHT.

ample

```
SET TEXT FONT -3, "STROKE"
SET TEXT HEIGHT 0.05
SET TEXT PATH "UP"
GRAPH TEXT AT 0.1,0.1 : "UP PATH"
SET TEXT PATH "LEFT"
GRAPH TEXT AT  0.7,0.9 : "LEFT PATH"
SET TEXT PATH "DOWN"
GRAPH TEXT AT 0.2,0.9 : "DOWN PATH"
SET TEXT PATH "RIGHT"
GRAPH TEXT AT 0.5,0.5 : "RIGHT PATH"
```

Output



ZK 4964-86

SET TEXT SPACE

The SET TEXT SPACE statement lets you change the spacing between characters drawn on an output device.

Format

SET TEXT SPACE *real-exp*

Syntax Rules

None.

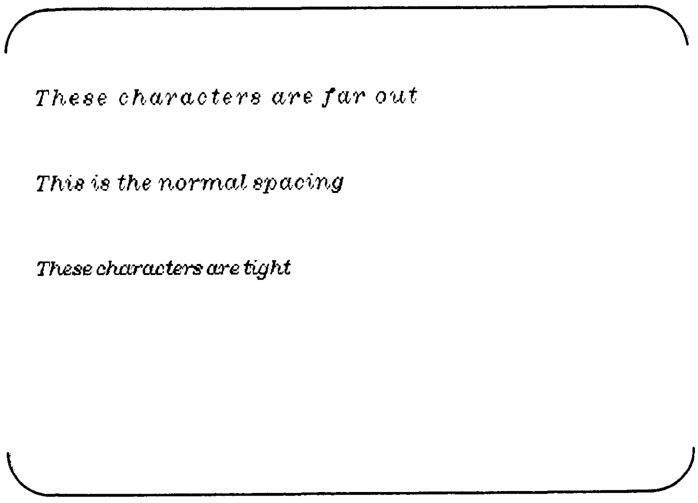
Remarks

1. Subsequent GRAPH TEXT statements use the spacing specified in *real-exp* until another SET TEXT SPACE statement is executed.
2. Space refers to the distance between adjacent characters. The distan is expressed as a fraction of the text height.
3. The default value for the spacing is 0, which produces adjacent characters as defined in the font design.
4. A positive value for *real-exp* inserts space between characters.
5. A negative value for *real-exp* causes characters to overlap.

xample

```
!+
!Set text height to 5/100 of the default window
!-
SET TEXT HEIGHT 0.05
SET TEXT FONT -16, "STROKE"
SET TEXT SPACE 0.09
GRAPH TEXT AT 0,0.8 : "These characters are far out"
SET TEXT SPACE 0
GRAPH TEXT AT 0,0.6 : "This is the normal spacing"
SET TEXT SPACE -0.09
GRAPH TEXT AT 0,0.4 : "These characters are tight"
```

Output



```
These characters are far out

This is the normal spacing

These characters are tight
```

ZK 4967-86

SET TRANSFORMATION

The SET TRANSFORMATION statement explicitly establishes the transformation to be used for subsequent interpretations of world coordinate points.

Format

SET TRANSFORMATION *int-exp*

Syntax Rules

Int-exp must be between 1 and 255.

Remarks

1. The current transformation number is set to the number you specify i *int-exp*. The default transformation is 1.
2. An implicit SET TRANSFORMATION is performed when a SET WINDOW or SET VIEWPORT statement is executed.
3. This transformation is used to display graphics output until a differen transformation is established in a SET TRANSFORMATION, SET WINDOW, or SET VIEWPORT statement.
4. A SET TRANSFORMATION statement is invalid within a picture definition.
5. A SET TRANSFORMATION statement implicitly sets the highest input priority to be the specified transformation. When conflicting viewports are defined, the transformation with the highest input priority is used to interpret input points. See also the SET INPUT PRIORITY statement.

xample

```
SET WINDOW , TRAN 1 : 0,10,0,10
SET VIEWPORT , TRAN 1 : 0,0.5,0,0.5
SET WINDOW , TRAN 2 : 500,1000,1,10
SET VIEWPORT ,TRAN 2 : 0.5,1,0.5,1
SET TRANSFORMATION 1
GRAPH AREA 2,8; 7,6; 3,1
SET TRANSFORMATION 2
GRAPH POINTS 800,2; 450,8; 475,8.5
    .
    .
    .
```

SET VALUE ECHO AREA

The SET VALUE ECHO AREA statement lets you specify the boundaries of the VALUE echo area.

Format

```
                          [ #dev-id        {   }  ]
SET VALUE ECHO AREA       [ , UNIT int-exp { : }  ]
                          [                {   }  ]
                      real-exp1, real-exp2, real-exp3, real-exp4
```

Syntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively.
2. When specified, *dev-id* must be the first clause listed and must be preceded by the number sign (#).
3. If one or more optional clauses are included, one colon (:) is required before *real-exp1.*

Remarks

1. The echo area is the portion of your screen where the prompt appear and where input can be supplied by a user.
2. The optional *dev-id* identifies the device for which you want to set th echo area. If no identification clause is included, VAX BASIC uses a default identification of #0. If an identification clause is included, the device specified must have been opened explicitly or opened by VAX BASIC as the default device.

3. Boundaries must be specified in device coordinates. To determine the possible values on a particular device, use the ASK DEVICE SIZE statement. To determine the current boundaries, use the ASK VALUE ECHO AREA statement. Appendix B lists the boundaries for VAXstations and VT125 and VT240 terminals.
4. The default echo area for VALUE input is device dependent. The default area is in effect until a SET VALUE ECHO AREA statement is executed.
5. The optional UNIT clause allows you to specify an alternative means of supplying the input. For instance, the position of points can be entered with mouse, or with the keyboard arrow keys. Each of these methods of data entry is a different *unit*. The default unit value for *int-exp* for each input type is 1; the default is used throughout this manual. For information about the units available on a particular device, see the hardware documentation for that device, as well as the VAX GKS documentation.
6. On devices that display the VALUE range limits, the size and justification of the characters cannot be altered. You should ensure that the VALUE echo area is appropriate for the VALUE prompt for the device you use.
7. Subsequent requests for each input type use the echo boundaries you supply until another SET VALUE ECHO AREA statement is executed.

xample

```
OPTION TYPE = EXPLICIT
DECLARE SINGLE left_1,right_1,bottom,top,input_value
OPEN "office-term" FOR GRAPHICS AS DEVICE #1
    .
    .
    .
ASK VALUE ECHO AREA #1 : left_1,right_1,bottom,top
!+
!Set the echo area to the top corner of the default area
!-
SET VALUE ECHO AREA #1 : left_1                  &
                        ,right_1                 &
                        ,(top-bottom)/2          &
                        ,top
LOCATE VALUE #1 : input_value
    .
    .
    .
```

SET VIEWPORT

The SET VIEWPORT statement allows you to change the section of NDC space to which an image in the world window is mapped.

Format

SET VIEWPORT *[* **, TRAN** *int-exp* : *] real-exp1, real-exp2, real-exp3, real-exp4*

Syntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively. Values must be within the range 0.0 to 1.0.
2. *Int-exp* must be between 1 and 255.

Remarks

1. The world viewport represents a portion of the abstract display surface known as the Normalized Device Coordinate space (NDC space). World coordinates are mapped to NDC space according to th specified boundaries in a SET VIEWPORT statement.
2. The default world viewport consists of the complete NDC space. Th boundaries at the start of program execution are 0,1,0,1 for the left, right, bottom, and top boundaries respectively.
3. Viewport boundaries must be within the range 0.0 to 1.0.
4. The SET VIEWPORT statement implicitly establishes the current transformation.
5. You can set a different world viewport for separate parts of your application, provided that you specify a different transformation number for each. For instance, the following statements set up the world viewport boundaries of 0,0.5,0.5,1 for transformation 1, and boundaries of 0.5,1,0.5,1 for transformation 2.

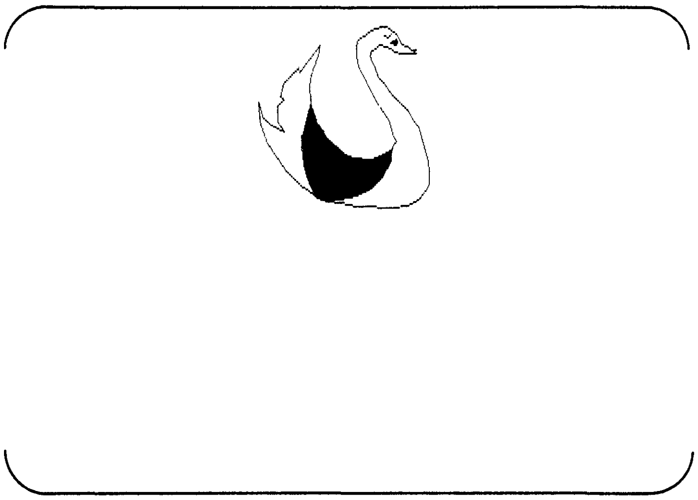
```
SET VIEWPORT , TRAN 1 : 0,0.5,0.5,1
SET VIEWPORT , TRAN 2 : 0.5,1,0.5,1
```

6. If you do not specify a transformation with the optional TRAN clause, the default transformation of 1 is assumed. At the start of program execution, transformation 1 defines the default window (0,1,0,1) and the default viewport (0,1,0,1).
7. All subsequent graphics output is interpreted using transformation *int-exp* until an alternative transformation is established in a SET TRANSFORMATION, SET WINDOW, or SET VIEWPORT statement.
8. A SET VIEWPORT statement implicitly sets the highest input priority to be the specified transformation. When conflicting viewports are defined, the transformation with the highest input priority is used to interpret input points. See also the SET INPUT PRIORITY statement.
9. The specified viewport boundaries remain in effect for transformation *int-exp* unless other boundaries are specified in a subsequent SET VIEWPORT statement referring to the same transformation.
10. A SET VIEWPORT statement is invalid within a picture.

xample

```
EXTERNAL PICTURE swan
    .
    .
    .
SET WINDOW , TRAN 2 : 0,100,0,100
!Map all of window to top right corner of NDC space
SET VIEWPORT , TRAN 2 : 0.5,1,0.5,1
    .
    .
    .
SET TRANSFORMATION 2
DRAW swan
    .
    .
    .
```

Output



ZK-4973-86

SET WINDOW

The SET WINDOW statement lets you change the boundary values (and therefore the coordinate measures) for the world window.

Format

SET WINDOW *[* **, TRAN** *int-exp :] real-exp1, real-exp2, real-exp3, real-exp4*

Syntax Rules

1. *Real-exp1, real-exp2, real-exp3,* and *real-exp4* must be floating-point expressions to represent the left, right, bottom, and top boundaries respectively.
2. *Int-exp* must be between 1 and 255.

Remarks

1. The default world window boundaries are 0,1,0,1 for the left, right, bottom, and top boundaries respectively.
2. When no transformation is specified, the transformation is set to transformation 1, the default.
3. The SET WINDOW statement implicitly establishes the current transformation.
4. The specified window boundaries remain in effect for transformation *int-exp* unless other boundaries are specified in a subsequent SET WINDOW statement referring to the same transformation.
5. All subsequent graphics output is interpreted using transformation *int-exp* until another transformation is established in a SET TRANSFORMATION, SET WINDOW, or SET VIEWPORT statement.
6. A SET WINDOW statement implicitly sets the highest input priority to be the specified transformation. When conflicting viewports are defined, the transformation with the highest input priority is used to transform input points. See also the SET INPUT PRIORITY statement.

7. When clipping is enabled with the SET CLIP "ON" statement, graphics images with world coordinate values exceeding the limits of the world window are not displayed.
8. When clipping is disabled with the SET CLIP "OFF" statement, images with world coordinate values that exceed the limits of the world window are displayed, provided that the points are also within the device window.
9. After a SET WINDOW statement, you may need to use the SET TEXT HEIGHT statement to increase the text height.
10. A SET WINDOW statement is invalid within a picture.

Example

```
!+
!Each of these output statements displays the center point
!-
GRAPH POINTS 0.5,0.5
CLEAR

SET WINDOW , TRAN 1: 0,100,0,100
GRAPH POINTS 50,50
CLEAR

SET WINDOW , TRAN 2 : 25,525,0,1000
GRAPH POINTS 275,500
CLEAR

SET WINDOW , TRAN 3 : -1,1,-1,1
GRAPH POINTS 0,0
CLEAR
SET WINDOW , TRAN 4 : -50,0,0,900
GRAPH POINTS -25,450
```

SHEAR

When used in a DRAW statement, the SHEAR function skews the x-coordinates of points specified in a PICTURE subprogram. SHEAR can also be used in the MAT statement to create a new matrix.

Format

In the DRAW Statement

DRAW *pic-name[(param-list)]* **WITH SHEAR***(angle) [* matrix2]...*

In the MAT Statement

MAT *matrix1* = **SHEAR***(angle) [* matrix2]...*

Syntax Rules

1. *Angle* can be in radians or degrees, depending on the option you specify with the OPTION ANGLE statement. The default is radians.
2. *Matrix1* and *matrix2* must be two-dimensional numeric arrays that are zero-based with upper bounds of 4 in both directions. VAX BASIC signals a compile-time error when the compiler detects a nonzero-based matrix; otherwise, a run-time error is signaled. Packed decimal arrays are invalid.
3. *Matrix2* can also be one of the valid transformation functions from the DRAW statement, including the following:
   - ROTATE
   - SCALE
   - SHIFT
   - TRANSFORM

Remarks

1. SHEAR can be used only on the right side of a MAT statement or as a transformation function in the DRAW statement.
2. Like other transformation functions, the SHEAR function affects coordinates displayed with PLOT and MAT PLOT statements within pictures. Similarly, input points accepted with GET and MAT GET statements within a picture definition are affected by the inverse of a SHEAR function specified on the DRAW statement that invokes the picture.
3. For an example and more information, see the DRAW statement in this chapter.

SHIFT

When used in a DRAW statement, the SHIFT function moves the coordinates of points specified in a PICTURE subprogram. SHIFT can also be used in the MAT statement to create a new matrix.

Format

In the DRAW Statement

DRAW *pic-name[(param-list)]* **WITH SHIFT** *(real-exp1, real-exp2) [* matrix2]...*

In the MAT Statement

MAT *matrix1* = **SHIFT** *(real-exp1, real-exp2) [* matrix2]...*

Syntax Rules

1. *Real-exp1* is applied to the x-coordinate of a point.
2. *Real-exp2* is applied to the y-coordinate of a point.
3. *Matrix1* and *matrix2* must be two-dimensional numeric arrays that are zero-based with upper bounds of 4 in both directions. VAX BASIC signals a compile-time error when the compiler detects a nonzero-based matrix; otherwise, a run-time error is signaled. Packed decimal arrays are invalid.
4. *Matrix2* can also be one of the valid transformation functions from the DRAW statement, including the following:
   - ROTATE
   - SCALE
   - SHEAR
   - TRANSFORM

Remarks

1. SHIFT can be used only on the right-hand side of a MAT statement or as a transformation function in the DRAW statement.
2. Negative values for *real-exp1* shift x-coordinates to the left; positive values shift coordinates to the right.
3. Negative values for *real-exp2* shift y-coordinates lower on the display surface; positive values shift coordinates up.
4. Like other transformation functions, the SHIFT function affects coordinates displayed with PLOT and MAT PLOT statements within pictures. Similarly, input points accepted with GET and MAT GET statements within a picture definition are affected by the inverse of a SHIFT function specified on the DRAW statement that invokes the picture.
5. For an example and more information, see the DRAW statement in this chapter.

TRANSFORM

The TRANSFORM function returns the cumulative transformation for all transformation clauses in current picture invocations.

Format

In the DRAW Statement

DRAW *pic-name[(param-list)]* **WITH TRANSFORM** *[* matrix2]...*

In the MAT Statement

MAT *matrix1* = **TRANSFORM** *[* matrix2]...*

Syntax Rules

1. *Matrix1* and *matrix2* must be two-dimensional numeric matrices that are zero-based with upper bounds of 4 in both directions. VAX BASIC signals a compile-time error when the compiler detects a nonzero-based matrix; otherwise, a run-time error is signaled. Packed decimal arrays are invalid.
2. *Matrix2* can also be one of the valid transformation functions from the DRAW statement, including the following:
   - SHIFT
   - SCALE
   - ROTATE
   - SHEAR

Remarks

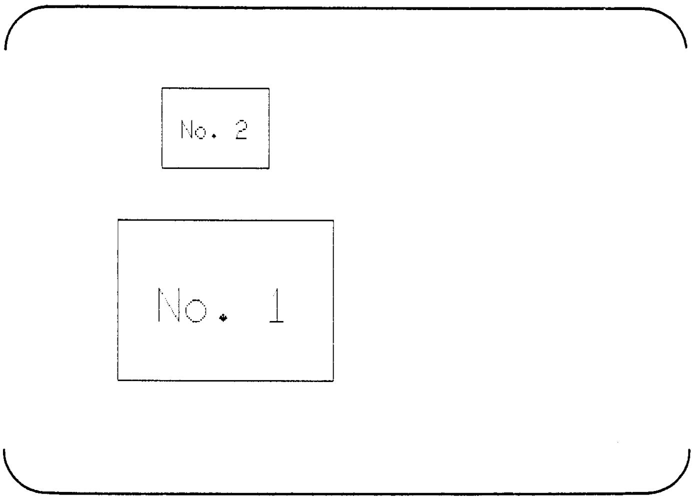
1. TRANSFORM can be used only on the right-hand side of a MAT statement or as a transformation function in the DRAW statement.
2. When used within picture invocations, TRANSFORM contains the cumulative matrix for all current transformations.
3. The current transformation is passed to any pictures by default. Therefore, if you use the TRANSFORM function inside a picture, the transformation in the DRAW statement will be applied *twice*.
4. If no picture has been invoked with a DRAW statement, TRANSFORM returns the identity matrix. The identity matrix has no effect on a graphics display.
5. Like other transformation functions, the TRANSFORM function affects coordinates displayed with PLOT and MAT PLOT statements within pictures. Similarly, input points accepted with GET and MAT GET statements within a picture definition are affected by the inverse of a TRANSFORM function specified on the DRAW statement that invokes the picture.
6. The TRANSFORM function can be used to adjust the text starting point and text height when text is drawn within a picture. The following example illustrates how to do this. Note that you cannot make adjustments for the text attributes for the ROTATE, SHEAR, and negative SCALE functions.
7. For an additional example and more information, see the DRAW statement in this chapter.

Example

```
PROGRAM transform_text
!+
!This program uses the TRANSFORM function to adjust the text height
!and text starting point for a GRAPH TEXT statement in a picture
!These attributes would not otherwise be affected by the
!transformation functions on a DRAW statement.
!-
OPTION TYPE = EXPLICIT
EXTERNAL PICTURE box(STRING,SINGLE,SINGLE)
DRAW box("No. 1",0.5,0.375) WITH SCALE(0.5) * SHIFT( 0.2, 0.1)
DRAW box("No. 2",0.5,0.375) WITH SCALE(0.25) * SHIFT(0.3, 0.6)
END PROGRAM
```

```
PICTURE box(STRING box_name, SINGLE x,y)
  OPTION TYPE = EXPLICIT
  DECLARE SINGLE height, new_height(1,4), old_height(1,4),  &
                 new_height_top, new_height_bottom,         &
                 new_x(1,4), new_y(1,4),                    &
                 old_x(1,4), old_y(1,4),                    &
          LONG counter
  PLOT LINES 0,0; 0,.75; 1,.75; 1,0; 0,0
  !+
  !Set the height of the letter to be one quarter
  !of the height of the box.
  !Get the current transformation for this picture
  !and transform the point.  Take the y coordinate of the result
  !and use it for the SET TEXT HEIGHT statement.
  !-
  old_height(1,counter) = 0.0 FOR counter = 0% TO 4%
  old_height(1,2) = .75/4
  old_height(1,4) = 1
  MAT new_height = old_height * TRANSFORM
  new_height_top = new_height(1,2)

  old_height(1,counter) = 0.0 FOR counter = 0% TO 4%
  old_height(1,2) = 0.0
  old_height(1,4) = 1
  MAT new_height = old_height * TRANSFORM
  new_height_bottom = new_height(1,2)

  old_x(1,counter) = 0.0 FOR counter = 0% TO 4%
  old_x(1,1) = x
  old_x(1,4) = 1
  MAT new_x = old_x * TRANSFORM

  old_y(1,counter) = 0.0 FOR counter = 0% TO 4%
  old_y(1,2) = y
  old_y(1,4) = 1
  MAT new_y = old_y * TRANSFORM

  SET TEXT HEIGHT (new_height_top - new_height_bottom)
  SET TEXT FONT 1%, "STROKE"
  SET TEXT JUSTIFY "CENTER", "HALF"
  GRAPH TEXT AT new_x(1,1), new_y(1,2) : box_name
END PICTURE
```

Output



ZK 4872 86

Appendix A

Calling VAX GKS Directly

VAX BASIC statements provide most of the features you will need to include graphics in your applications; however, you may choose to access additional graphics functionality by calling VAX GKS routines directly. In all instances, when VAX BASIC graphics statements are available, it is recommended that you use them rather than make direct calls yourself.

ι.1 Introduction

VAX GKS operates in one of four states. The states are

- Closed
- Open
- At least one device open
- At least one device active

VAX BASIC controls the VAX GKS states for you by default. For instance, if you try to display output on a device when none is open and active, VAX BASIC opens and activates the default device for you. Similarly, if you try to close a device that you have not opened, VAX BASIC ignores your request to close it.

You must ensure that VAX GKS is in the correct operating state by using VAX GKS control routines in either of the following situations:

- When a program includes no VAX BASIC graphics statements at all, but does include direct calls to VAX GKS routines
- When a program makes direct calls to VAX GKS routines *before* executing any VAX BASIC graphics statements

When VAX GKS is not in the correct state, an error is issued and program execution terminates. Consult the VAX GKS documentation for information about particular routines and the related error messages.

VAX BASIC sets up many preferred defaults that do not always agree witl the VAX GKS defaults for a particular routine. The reference section in thi manual lists the defaults for each VAX BASIC graphics statement. Most of the VAX BASIC defaults that are device specific are set up when a VA) BASIC graphics output statement is executed or when an OPEN . . . FOR GRAPHICS statement is executed. If you have already used VAX GKS routines in your program, other defaults may be in place. The following list includes items that you should be aware of when you make direct call to VAX GKS.

- If you want VAX BASIC graphics defaults, be sure to use the OPEN . . . FOR GRAPHICS statement, or include a graphics output statement *before* you make any direct calls to VAX GKS.
- If you want VAX GKS defaults, you should open a device with explic calls to GKS$OPEN_WS and GKS$ACTIVATE_WS to override the VAX BASIC defaults. Note that when you use VAX GKS control statements, VAX GKS should be opened with an error file of "NLA0" This is important for VAX GKS error handling. You should be aware of the different defaults and use caution when mixing VAX BASIC graphics statements with direct calls to VAX GKS because program output can be unpredictable.
- In VAX BASIC, transformation 0 is reserved and any references to transformation 0 in program statements are invalid. When you make direct calls to VAX GKS routines, transformation 0 is accessible. Nonetheless, you *cannot* change the priority of transformation 0 whe calling VAX GKS routines from VAX BASIC programs.
- The default text expansion factor of 1 is the height-to-width ratio defined in the font design. VAX BASIC maintains this "normal" ratio regardless of the current window boundaries. That is, whenever a new transformation becomes current, VAX BASIC implicitly adjusts the text expansion factor according to the ratio of the current windov

width to the current window height. However, VAX GKS produces a "normal" text expansion factor only when the world coordinate units for the x- and y-axes are the same. When you make direct calls to VAX GKS, you must make your own adjustments to the expansion factor when necessary.

The following sections show you how to call VAX GKS routines directly and provide examples of such calls. The VAX GKS routines are described fully in the VAX GKS documentation.

Like other system routines, each VAX GKS routine has an entry point (the routine name) and an argument list. It may also return values to the program that calls it. To call VAX GKS routines directly, you should follow these steps:

1. Declare the desired VAX GKS routines with the EXTERNAL statement
2. Declare all arguments with correct data types and passing mechanisms
3. Call the desired routines
4. Check the return status of the call

\.2 Declaring VAX GKS Routines

As with other system routines, you should declare VAX GKS routines as external functions rather than subprograms. To do this, use the EXTERNAL statement as shown in the following examples. Note that if you call a VAX GKS routine as a subprogram, you cannot check the status of the call. The following partial program illustrates how to make a simple call to a VAX GKS routine.

Example

```
PROGRAM screen_makeup
  OPTION TYPE = EXPLICIT
  !+
  !Declare the routine as a function
  !-
  EXTERNAL LONG FUNCTION GKS$UPDATE_WS(LONG BY REF,LONG BY REF)
  EXTERNAL LONG CONSTANT STS$K_SUCCESS
  EXTERNAL SUB LIB$SIGNAL(LONG BY VALUE)
  !+
  !Declare the arguments
  !-
  DECLARE LONG dev_id, flag_value ,return_status
  dev_id = 1%
   .
   .
   .
  !+
  !Call GKS$UPDATE_WS---this gives you control of the device
  !viewport (VAX BASIC performs this function by default when you define more t
  !one device viewport and use VAX BASIC graphics statements)
  !-
  return_status = GKS$UPDATE_WS(dev_id,flag_value)
  !+
  !Check the status
  !-
  IF (return_status AND 1%) = STS$K_SUCCESS
     THEN GOTO device_setup
     ELSE CALL LIB$SIGNAL(return_status)
          EXIT PROGRAM
  END IF
   .
   .
   .
```

Arguments that are passed to VAX GKS functions must be listed in the order shown in the function descriptions in the VAX GKS documentation. Arguments are listed in the VAX GKS documentation as being write or modify, or read only. When VAX BASIC encounters arguments of the wrong data type, an error is signaled at compile time. VAX GKS expects the following data types and passing mechanisms:

- Integer arguments as 32-bit longwords passed by reference
- Real numbers in single-precision floating-point format passed by reference
- Character strings passed by descriptor
- Arrays passed either by reference or by descriptor, depending on the function

See the VAX GKS documentation to determine the rules governing the arguments you require for a particular function.

\.3 Useful VAX GKS Routines

Not all VAX GKS functionality is available with VAX BASIC statements. It is beyond the scope of this manual to illustrate the full extent of the additional functionality;. instead, three examples are provided here of functionality you can access by making direct calls to VAX GKS. The first example shows how to change the device echo type default (VAX BASIC offers a fixed default echo type). The second example show you how to use VAX GKS cell arrays. The third example illustrates the use of attribute bundles.

\.3.1 Changing the Echo Type

The GKS$INIT_LOCATOR function allows you to take advantage of additional echo types for POINT input on VT125 and VT240 terminals. VAX BASIC supplies the VAX GKS default echo type, as illustrated in Chapter 7. To change this default, you must supply the alternative echo type as an argument to the GKS$INIT_LOCATOR routine.

This example changes the echo type to 6. On VT125 and VT240 terminals, echo type 6 for POINT input consists of a display of the device coordinates of the point selected by a user. (Echo types are device dependent; consult the VAX GKS documentation for details.)

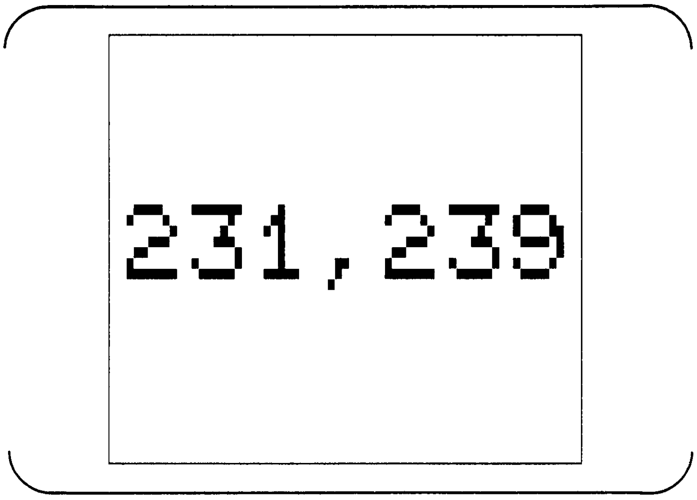
With echo type 6, the position of the point is not displayed, only the coordinate values. The user can change the device coordinates with the arrow keys. The VAX GKS routine returns the equivalent world coordinates of this same point into two of the arguments you supply with the function call. The example displays the device coordinates and the world coordinates of the same point supplied by the user.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL LONG FUNCTION GKS$INIT_LOCATOR                                &
        (LONG,LONG,SINGLE,SINGLE,LONG,LONG,                            &
         SINGLE,ANY,LONG)
EXTERNAL LONG CONSTANT STS$K_SUCCESS
DECLARE SINGLE echo_area(3)
DECLARE LONG tran_id, dev_id, return_status                            &
                input_type_val,echo_type,data_size
DECLARE SINGLE loc_x,loc_y,user_x,user_y
!+
!This record is required but ignored by GKS
!-
RECORD LOC_DATA_REC
     LONG loc_data
END RECORD

DECLARE LOC_DATA_REC rec1
OPEN "" FOR GRAPHICS AS DEVICE #1

dev_id = 1
input_type_val = 1
echo_area(0) = 240
echo_area(1) = 767
echo_area(2) = 0
echo_area(3) = 479
!+
!Initial point in world coords
!User is presented with device coords of this point
!-
loc_x = 0.5
loc_y = 0.5
!+
!Set the new echo type
!-
echo_type = 6%
data_size = 0%

return_status = GKS$INIT_LOCATOR(dev_id,input_type_val                   &
                ,loc_x,loc_y,tran_id,echo_type,echo_area()               &
                ,rec1,data_size)
```

```
IF (return_status AND 1%) = STS$K_SUCCESS
          THEN LOCATE POINT user_x,user_y
          ELSE EXIT PROGRAM return_status
END IF
!+
!Display world coordinates of user_x, user_y
!-
GRAPH TEXT AT 0,0.95 : "World coordinates: " + STR$(user_x)
GRAPH TEXT AT 0,0.90 : STR$(user_y)
END PROGRAM return_status
```

Output



ZK-4987-86

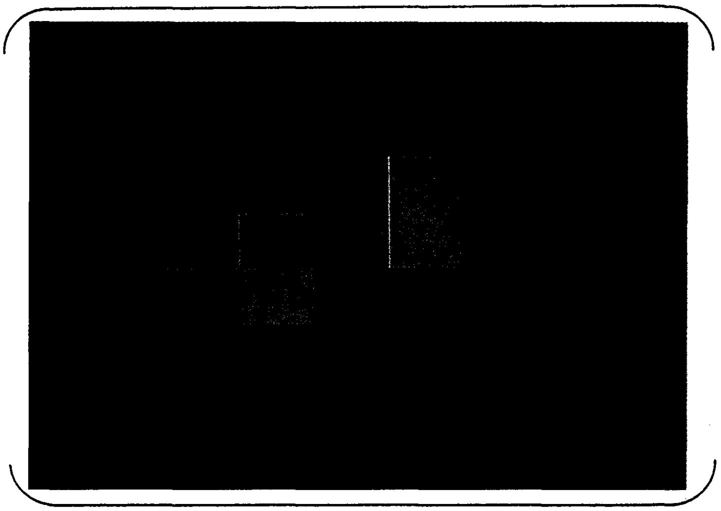
\.3.2 Using VAX GKS Cell Arrays

The following example shows how to access the VAX GKS cell array feature. The example divides a rectangular area into cells and displays each cell in a specific color. To specify color, the example defines an array and supplies color index values as data to the array.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL LONG FUNCTION GKS$CELL_ARRAY(SINGLE,SINGLE,SINGLE,SINGLE,        &
                       LONG,LONG,LONG,LONG,LONG DIM (,))
EXTERNAL LONG CONSTANT STS$K_SUCCESS
DECLARE SINGLE first_x, first_y, second_x, second_y
DECLARE LONG no_rows, no_cols, start_row, start_col,                     &
             counter, inner,return_status
DIM LONG colour(3,2)
OPEN "" FOR GRAPHICS AS DEVICE #1
!+
!Fill the array with color index values
!-
DATA 3,2,0, 1,3,2, 0,2,2, 3,1,1
FOR counter = 0% TO 3%
  FOR inner = 0% TO 2%
    READ colour(counter,inner)
  NEXT inner
NEXT counter

first_x = 0.2
first_y = 0.3
second_x = 0.9
second_y = 0.7
start_row = 0%
start_col = 0%
no_rows = 4%
no_cols = 3%
return_status = GKS$CELL_ARRAY(first_x,first_y,second_x,second_y         &
                    ,start_row,start_col,no_rows,no_cols,colour(,))
IF (return_status AND 1%) = STS$K_SUCCESS
   THEN
   CLOSE DEVICE #1
END IF
END
```

Output



.3.3 Accessing Attribute Bundles

When you call VAX GKS directly, you can change attributes in groups, or *bundles*. VAX BASIC provides the capability to change each of the attributes of graphics objects individually. For instance, you can change the line style, the point color, and the text size. How to change the attributes is discussed in Chapter 3. With direct calls to VAX GKS routines, you can specify a set of predefined attribute values for an object.

VAX GKS stores predefined attribute values for each object in a construct known as a *bundle table*. The bundle table consists of several sets of predefined attribute values; you specify an index in the bundle table to select a given set of attribute values for an object. Each hardware device has its own set of bundle tables. You can select from several sets of predefined values to achieve the image you want. For details of the bundle tables for a particular device, see the VAX GKS documentation.

The attributes are governed by Attribute Source Flags (ASFs). Each ASF can have a value of 0 for bundled, or 1 for individual. To specify an index to the bundle tables, you must call the routine GKS$SET_ASF to set selected ASFs to the value 0 for bundled. By default, each of the ASFs is set to 1. You must pass an array of 13 values (0 or 1 each) to the GKS$SET_ASF routine. The order of the flags in the bundle table is as follows:

1. Line type
2. Line width scale factor
3. Line color index
4. Point type
5. Point size scale factor
6. Point color index
7. Text font and precision
8. Character expansion factor
9. Character spacing
10. Text color index
11. Fill area interior style
12. Fill area style index
13. Fill area color index

When program execution begins, all of these flags are set to individual (1). If no flag values are changed, then graphics objects are displayed with the individual VAX GKS default values.

Note that VAX BASIC attribute statements do not change the ASFs; therefore, setting an ASF to bundled effectively disables some of the VAX BASIC SET statements and invalidates the corresponding ASK statements.

The following example displays sample text using the initial text attributes. The example then sets a selection of the flags to 0 and calls the GKS$SET_ASF function specifying which bundle of attributes to use. The text is displayed again showing the bundled text attributes.

Example

```
OPTION TYPE = EXPLICIT
EXTERNAL LONG FUNCTION GKS$SET_ASF(LONG DIM () BY REF)
DECLARE LONG flags(1 TO 13)
DECLARE LONG counter, return_status
EXTERNAL LONG CONSTANT STS$K_SUCCESS
!+
!Initialize array to 1, for individual
!-
flags(counter) = 1 FOR counter = 1 TO 13
!+
!Display text with individual defaults
!-
GRAPH TEXT AT 0.0,0.8 : "Text attributes set to individual."
!+
!Change flags 7-10 to 0 for bundles
!-
flags(counter) = 0 FOR counter = 7% TO 10%
!+
!Set the ASFs with the new array
!-
return_status = GKS$SET_ASF(flags())
!+
!Check the return status
!If success, display text with bundled attributes
!-
IF (return_status AND 1%) = STS$K_SUCCESS
   THEN GRAPH TEXT AT 0.0,0.5 : "Attributes from a bundle table."
   ELSE GRAPH TEXT AT 0.1,0.4 : "Error in GKS$SET_ASF"
END IF
END
```

Appendix B

Device Specifications

This appendix lists information specific to the following DIGITAL devices:

- VT125 and VT240 terminals
- VAXstations I, II, and II/GPX

The initial and default values listed here refer to the selections available with VAX BASIC graphics statements. These devices support additional features (such as alternative echo styles) that are not available with VAX BASIC graphics statements. For more information, see the VAX GKS documentation. Information about other devices can also be found in the VAX GKS documentation, as well as in the documentation for the individual device.

Section B.3 of this appendix provides a chart that lists and illustrates values for the color intensities of red, green, and blue that can be used on VT125 and VT240 terminals and VAXstations.

B.1 VT125 and VT240 Terminals

This section describes the information needed to use VAX BASIC graphics with monochrome and color VT125 and VT240 terminals.

B.1.1 Device Types

The following device types are valid:

GKS$WSTYPE	Device
10	VT125 monochrome, output only
11	VT125 with color option
12	VT125 monochrome
13	VT240 with color option
14	VT240 monochrome

B.1.2 Text Fonts

VT125 and VT240 terminals support one hardware font and 23 software fonts. These fonts are illustrated in Chapter 4 of this manual.

B.1.3 Pattern Values

The following table lists the values for the area style index when the area style is set to PATTERN.

Style Index	Appearance
1	Mixes colors 1 and 2
2	Mixes colors 2 and 3
3	Mixes colors 3 and 1
4	Mixes colors 0 and 1
5	Mixes colors 0 and 2
6	Mixes colors 0 and 3

B.1.4 Hatch Values

The following table lists the values for the area style index when the area style is set to HATCH. Some of these indices are illustrated in Chapter 3.

Style Index	Appearance
1	Cross hatches
2	Horizontal lines at 45 degrees
3	Horizontal lines at –45 degrees
4	Horizontal lines
5	Vertical lines
6	Horizontal lines at 45 degrees—sparse
7	Horizontal lines at –45 degrees—sparse
8	Horizontal lines—sparse
9	Vertical lines—sparse
10–19	Varying density hatches[1]
33–126	Hatching with the corresponding ASCII character

[1] A varying density hatch style is a mixture of pixels of a given color and white pixels, which gives the appearance of different shades. For hatch styles 10 through 19, the hatches progress from lighter to darker; 10 being the lightest and 19 being the darkest.

.1.5 Device Coordinates

The VT125 device coordinates have the range ([0,767] x [0,479]); the VT240 device coordinates have the range ([0,799] x [0,479]).

.1.6 Input Constructs

On VT125 and VT240 terminals, you press the RETURN key to indicate the completion of input. You use the arrow keys to move the cursor. For POINT, MULTIPOINT, and VALUE input, you can also use the PF3 key to move the cursor a shorter distance and the PF4 key to move the cursor a greater distance.

B.1.6.1 POINT Input

The default values for POINT input on VT125 and VT240 terminals are as follows:

Input Construct	Default Value
Unit	1
Initial cursor position	(0.5, 0.5) in world coordinates
Echo area	[(0,479.0) x (0,479.0)] in device coordinates

B.1.6.2 MULTIPOINT Input

When you enter MULTIPOINT input, press the space bar to enter each point and press RETURN when the series of points is complete. A line is displayed that joins successive points in the series of points entered. The DELETE key eliminates the last point entered.

The default values for MULTIPOINT input on VT125 and VT240 terminals are as follows:

Input Construct	Default Value
Unit	1
Initial number of points	0
Echo area	[(0,479.0) x (0,479.0)] in device coordinates

.1.6.3 CHOICE Input

The default values for CHOICE input on VT125 and VT240 terminals are as follows:

Input Construct	Default Value
Unit	1
VT125 echo area	[(513.0,767.0) x (0,479.0)] in device coordinates
VT240 echo area	[(533.0,767.0) x (0,479.0)] in device coordinates
Initial number of choices	5
Initial choice	1
Default menu	"CHOICE 1", "CHOICE 2", and so on through "CHOICE 5"

.1.6.4 STRING Input

The default values for STRING input on VT125 and VT240 terminals are as follows:

Input Construct	Default Value
Unit	1
Initial string	Null
VT125 echo area	[(513.0,767.0) x (0,479.0)] in device coordinates
VT240 echo area	[(533.0,767.0) x (0,479.0)] in device coordinates

B.1.6.5 VALUE Input

The default values for VALUE input on VT125 and VT240 terminals are a follows:

Input Construct	Default Value
Unit	1
Initial range	0.0—1.0
Initial value	0.5
VT125 echo area	[(513.0,767.0) x (0,479.0)] in device coordinates
VT240 echo area	[(533.0,767.0) x (0,479.0)] device coordinates

B.1.7 Color Index Values

The default color index values for monochrome terminals are as follows:

Index	Color	Red Intensity	Green Intensity	Blue Intensity
0	Background	0.0	0.0	0.0
1	Normal	0.0	1.0	0.0
2	Dark	1.0	0.0	0.0
3	Light	1.0	1.0	1.0

The default color index values for color terminals are as follows:

Index	Color	Red Intensity	Green Intensity	Blue Intensity
0	Black	0.0	0.0	0.0
1	Green	0.0	1.0	0.0
2	Red	1.0	0.0	0.0
3	Blue	0.0	0.0	1.0

;.2 VAXstations

This section describes the information needed to use VAX BASIC graphics statements with VAXstations I, II, and II/GPX.

.2.1 Device Types

VAXstations I, II, and II/GPX are all device type 41.

.2.2 Windowing Capabilities

VAX BASIC graphics does not support calls to the VAXstation windowing software; therefore, you should not use VAX BASIC graphics statements and calls to the window manager products in the same program.

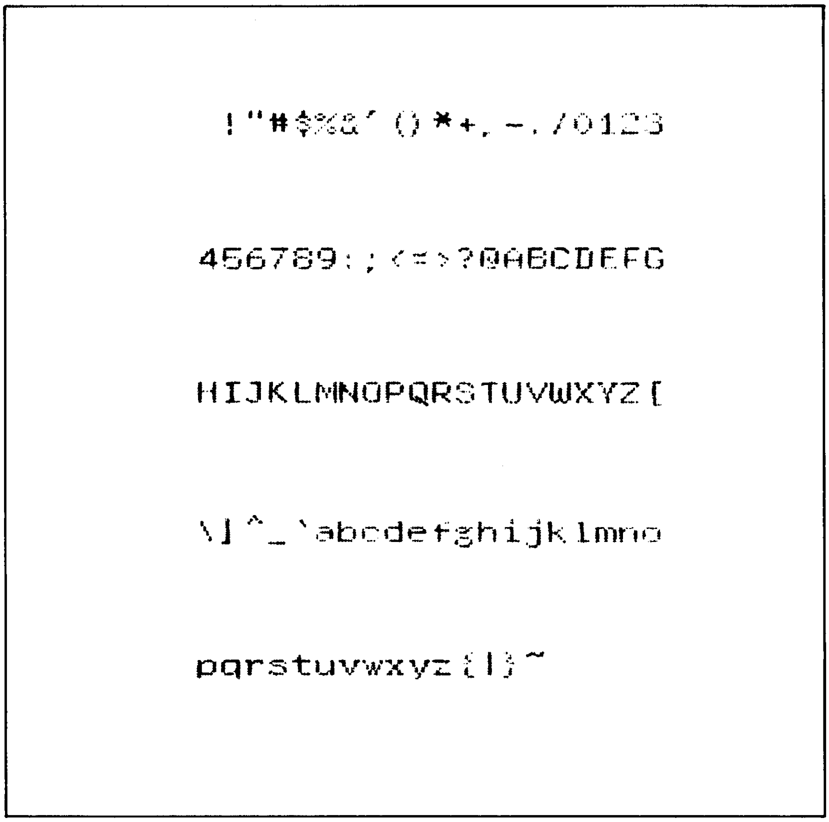
Output on VAXstations is displayed on a new window and the output display disappears when the new window is closed. There are various ways to keep the output display active. For instance, you can include SLEEP statements in your programs.

When performing input, VAX BASIC positions the display window as defined by the input echo area (device coordinates). When generating output, VAX BASIC positions the display window as defined by the current device viewport.

.2.3 Text Fonts

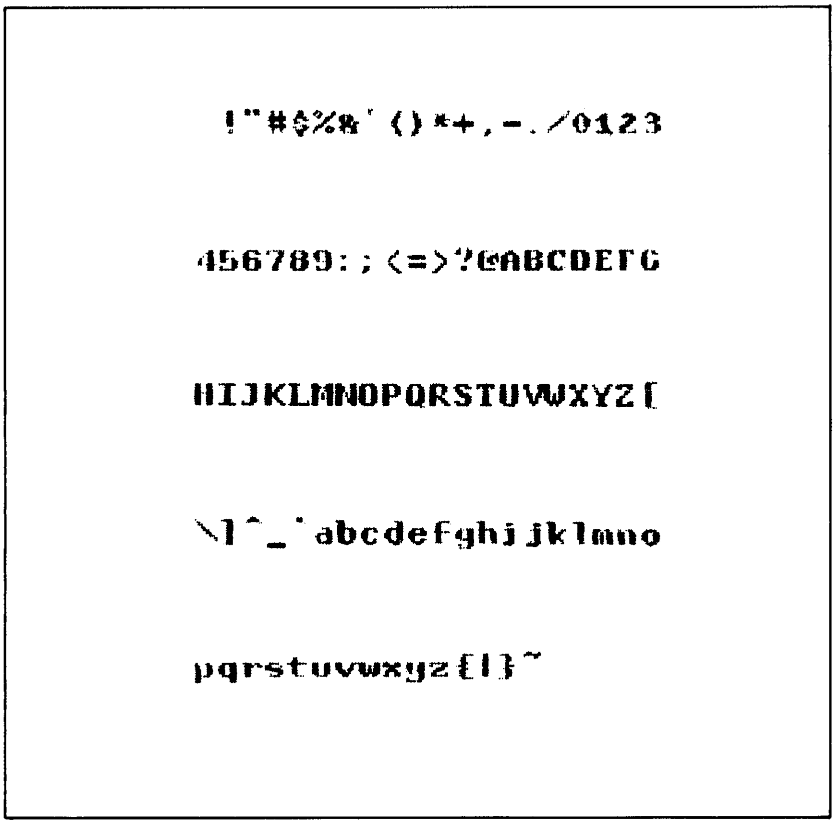
VAXstations support the four hardware fonts illustrated in the following figures.

Figure B–1: VAXstation Font Number 1



ZK-3062-84

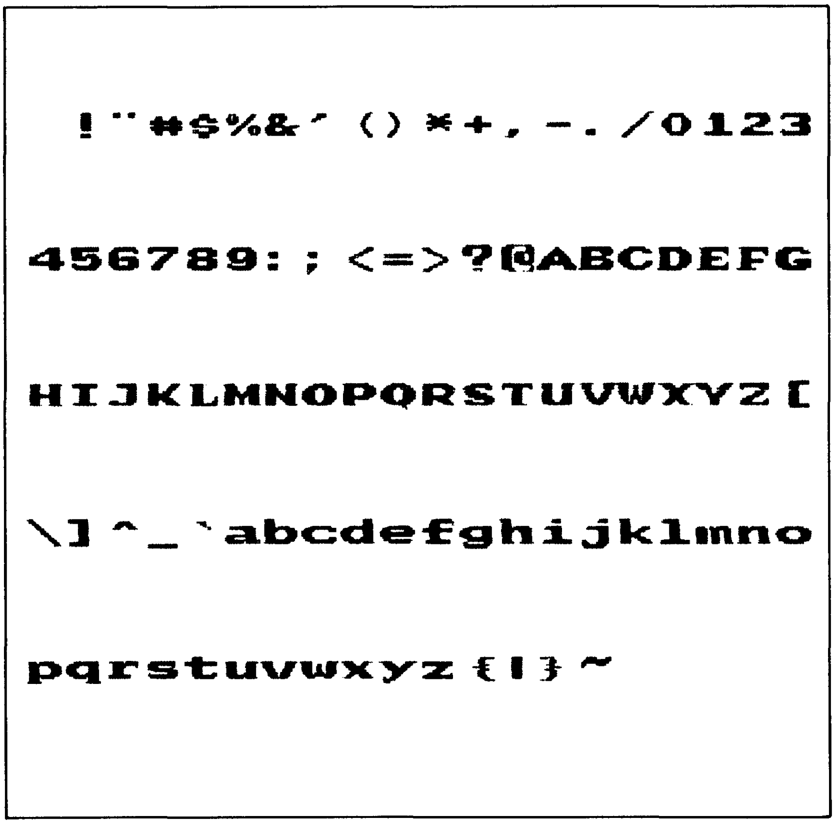
Figure B–2: VAXstation Font Number –1



```
 !"#$%&'()*+,-./0123

456789:;<=>?@ABCDEFG

HIJKLMNOPQRSTUVWXYZ[

\]^_`abcdefghijklmno

pqrstuvwxyz{|}~
```

ZK-3063-84

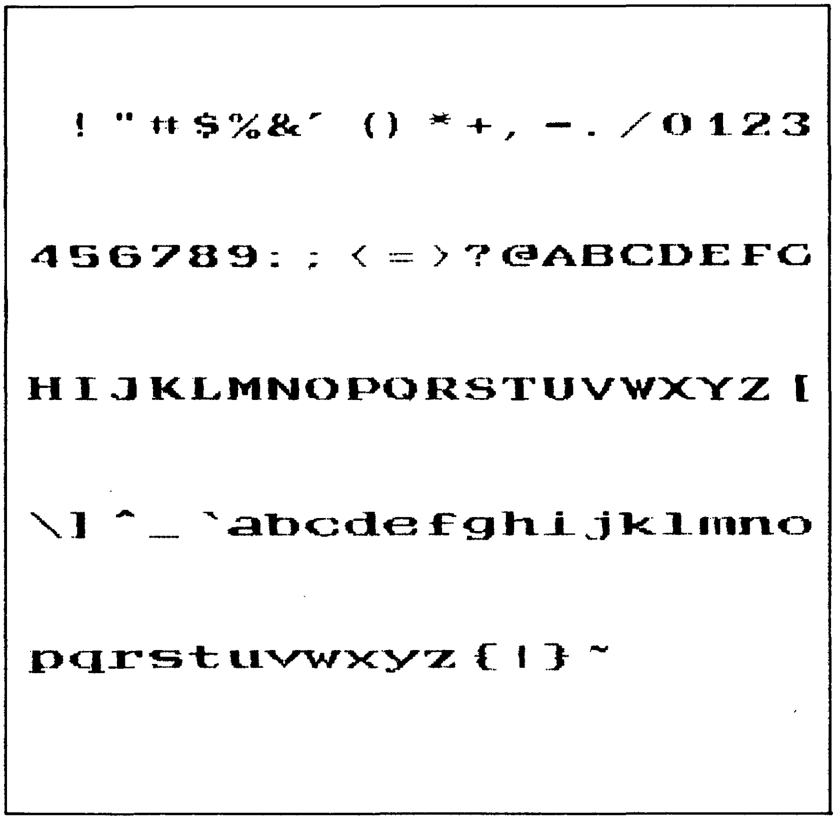
Figure B–3: VAXstation Font Number –2



```
!"#$%&'()*+,-./0123
456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[
\]^_`abcdefghijklmno
pqrstuvwxyz{|}~
```

ZK-3064-84

Figure B–4: VAXstation Font Number –3


! " # $ % & ' ( ) * + , - . / 0 1 2 3
4 5 6 7 8 9 : ; < = > ? @ A B C D E F G
H I J K L M N O P Q R S T U V W X Y Z [
\ ] ^ _ ` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~

ZK-3065-84

2.4 Pattern Values

The following table lists the values for the area style index when the area style is set to PATTERN. The default color index values are assumed; however, you can change these with the SET COLOR MIX statement.

Style Index	Appearance
1	Black light diagonally woven pattern (25%)
2	Red light diagonally woven pattern (25%)
3	Green light diagonally woven pattern (25%)
4	Blue light diagonally woven pattern (25%)
5	Cyan light diagonally woven pattern (25%)
6	Magenta light diagonally woven pattern (25%)
7	Yellow light diagonally woven pattern (25%)
8	Black darker diagonally woven pattern (50%)
9	Red darker diagonally woven pattern (50%)
10	Green darker diagonally woven pattern (50%)
11	Blue darker diagonally woven pattern (50%)
12	Cyan darker diagonally woven pattern (50%)
13	Magenta darker diagonally woven pattern (50%)
14	Yellow darker diagonally woven pattern (50%)
15	Black darkest diagonally woven pattern (75%)
16	Red darkest diagonally woven pattern (75%)
17	Green darkest diagonally woven pattern (75%)
18	Blue darkest diagonally woven pattern (75%)
19	Cyan darkest diagonally woven pattern (75%)
20	Magenta darkest diagonally woven pattern (75%)
21	Yellow darkest diagonally woven pattern (75%)
22	Black horizontal brick pattern
23	Red horizontal brick pattern
24	Green horizontal brick pattern
25	Blue horizontal brick pattern
26	Cyan horizontal brick pattern
27	Magenta horizontal brick pattern
28	Yellow horizontal brick pattern
29	Black vertical brick pattern
30	Red vertical brick pattern

Style Index	Appearance
31	Green vertical brick pattern
32	Blue vertical brick pattern
33	Cyan vertical brick pattern
34	Magenta vertical brick pattern
35	Yellow vertical brick pattern
36	Black brick pattern at –45 degrees
37	Red brick pattern at –45 degrees
38	Green brick pattern at –45 degrees
39	Blue brick pattern at –45 degrees
40	Cyan brick pattern at –45 degrees
41	Magenta brick pattern at –45 degrees
42	Yellow brick pattern at –45 degrees
43	Black brick pattern at 45 degrees
44	Red brick pattern at 45 degrees
45	Green brick pattern at 45 degrees
46	Blue brick pattern at 45 degrees
47	Cyan brick pattern at 45 degrees
48	Magenta brick pattern at 45 degrees
49	Yellow brick pattern at 45 degrees
50	Black finely woven grid
51	Red finely woven grid
52	Green finely woven grid
53	Blue finely woven grid
54	Cyan finely woven grid
55	Magenta finely woven grid
56	Yellow finely woven grid
57	Black sparsely woven grid
58	Red sparsely woven grid
59	Green sparsely woven grid
60	Blue sparsely woven grid

Style Index	Appearance
61	Cyan sparsely woven grid
62	Magenta sparsely woven grid
63	Yellow sparsely woven grid
64	Black downward scales (fish-like)
65	Red downward scales (fish-like)
66	Green downward scales (fish-like)
67	Blue downward scales (fish-like)
68	Cyan downward scales (fish-like)
69	Magenta downward scales (fish-like)
70	Yellow downward scales (fish-like)
71	Black upward scales
72	Red upward scales
73	Green upward scales
74	Blue upward scales
75	Cyan upward scales
76	Magenta upward scales
77	Yellow upward scales
78	Black rightward scales
79	Red rightward scales
80	Green rightward scales
81	Blue rightward scales
82	Cyan rightward scales
83	Magenta rightward scales
84	Yellow rightward scales

Style Index	Appearance
85	Black leftward scales
86	Red leftward scales
87	Green leftward scales
88	Blue leftward scales
89	Cyan leftward scales
90	Magenta leftward scales
91	Yellow leftward scales
91–203	Increasing densities, incrementing first by color, starting at 1/16 density, incrementing by 1/16, up to 15/16 density

.2.5 Hatch Values

The following table lists the values for the area style index when the area style is set to HATCH.

Style Index	Appearance
–1	Cross hatching
–2	Diagonal lines at 45 degrees (50%)
–3	Diagonal lines at –45 degrees (50%)
–4	Horizontal lines (50%)
–5	Vertical lines (50%)
–6	Diagonal lines at 45 degrees—sparse (12.5%)
–7	Diagonal lines at –45 degrees—sparse (12.5%)
–8	Horizontal lines—sparse (12.5%)
–9	Vertical lines—sparse (12.5%)
–10	Diagonal lines at 45 degrees (25%)
–11	Diagonal lines at –45 degrees (25%)
–12	Horizontal lines (25%)
–13	Vertical lines (25%)
–14	Diagonal lines at 45 degrees—sparse (6.25%)

Style Index	Appearance
-15	Diagonal lines at -45 degrees—sparse (6.25%)
-16	Horizontal lines—sparse (6.25%)
-17	Vertical lines—sparse (6.25%)
-18	Diagonal lines at 45 degrees (75%)
-19	Diagonal lines at -45 degrees (75%)
-20	Horizontal lines (75%)
-21	Vertical lines (75%)
-22	Diagonal lines at 45 degrees—sparse (50%)
-23	Diagonal lines at -45 degrees—sparse (50%)
-24	Horizontal lines—sparse (50%)
-25	Vertical lines—sparse (50%)
-26	Diagonal lines at 45 degrees—sparse (75%)
-27	Diagonal lines at -45 degrees—sparse (75%)
-28	Horizontal lines—sparse (75%)
-29	Vertical lines—sparse (75%)
-30	Horizontal lines—very fine (50%)
-31	Vertical lines—very fine (50%)
-32	Finely woven grid
-33	Sparsely woven grid

B.2.6 Device Coordinates

The VAXstation device coordinates have the range ([0,0.3319027] x [0,0.2738526]), in meters.

The VAXstation handler divides the device coordinate range into raster units to establish the pixels actually used by the physical device. The VAXstation raster unit range is ([0,1011] x [0,834]).

.2.7 Input Constructs

On a VAXstation, you can enter input with the keyboard or the mouse. With the keyboard, you press the RETURN key to indicate the completion of input. You use the arrow keys to move the cursor. For POINT, MULTIPOINT, and VALUE input, you can use the PF3 key to move the cursor a shorter distance and the PF4 key to move the cursor a greater distance.

For all input types except STRING, pressing the leftmost mouse button enters the input value. The middle button on the mouse causes the input to be canceled. You cannot use the keyboard to enter or cancel input. For STRING input, use the RETURN key to enter input and the CTRL/U key to cancel input. To begin entering MULTIPOINT input, you press and release the leftmost button on the mouse, and then move the mouse as desired.

The input units for VAXstations are as follows:

Input Types	Units
POINT	1
MULTIPOINT	1
VALUE	1
CHOICE	1
STRING	1, 2

For all input types, you specify the echo area in device coordinates. The device coordinate system origin for VAXstations is in the lower left corner of the display surface. By default, the VAXstation handler uses an echo area whose lower left corner is located in the lower left corner of the display surface (the origin of the device coordinate system).

If, during input, you move the mouse outside the echo area, mouse tracking is disabled until the mouse is once again within the echo area.

When you request input, the VAXstation handler pops the workstation viewport to the top of any other overlapping viewports. Viewport appearance before and after the input request is unchanged.

The sections that follow briefly describe each input type and show initial values for each.

B.2.7.1 POINT Input

The default values for POINT input on VAXstations are as follows:

Input Construct	Default Value
Unit	1
Initial position	(0.5, 0.5) in world coordinates
Echo area	[(0,0.3286231) x (0,0.2695890)] in device coordinates (meters)

If the user attempts to enter a point outside of of the echo area, the VAXstation handler ignores the entry.

B.2.7.2 MULTIPOINT Input

The default values for MULTIPOINT input on VAXstations are as follows:

Input Construct	Default Value
Unit	1
Initial number of points	0
Echo area	[(0,0.3286231) x (0,0.2695890)] in device coordinates (meters)

B.2.7.3 CHOICE Input

The default values for CHOICE input on VAXstations are as follows:

Input Construct	Default Value
Unit	1
Echo area	[(0,0.02) x (0,0.02)] in device coordinates (meters)
Initial number of choices	2
Default strings	"YES", "NO"

;.2.7.4 STRING Input

The default values for STRING input on VAXstations are as follows:

Input Construct	Default Value
Unit	1
Initial string	Null
Echo area	[(0,0.2) x (0,0.01)] in device coordinates (meters)

STRING Input Unit 1

STRING input unit 1 returns a DEC Multinational text string to the calling program.

An underline cursor marks the current position within the string. The DELETE key deletes the previous character from the string and erases the character from the display. The left and right arrow keys change the editing position within the string. Pressing CTRL/A toggles between insert and overlay mode.

You enter STRING input unit 1 by pressing the RETURN key, and you cancel input by pressing CTRL/U.

STRING Input Unit 2

The string logical input unit 2 returns an SMG Encoded Key value. The input request for this unit ends as soon as the user presses a key on the keyboard. For more information on STRING input unit 2, refer to the documentation for the SMG Run-Time Library Routines.

;.2.7.5 VALUE Input

The echo area must be at least 0.03411 meters wide by 0.01476 meters high, so that there is sufficient room to display the digital value representation of the current value. If the echo area is too small, an error is generated.

The default values for VALUE input on VAXstations are as follows:

Input Construct	Default Value
Unit	1
Initial range	0.0—1.0
Initial value	0.5
Echo area	[(0,0.127) x (0,0.0254)] in device coordinates (meters)

B.2.8 Color Index Values

The following table lists the default red, green, and blue intensity associated with each color index value on monochrome VAXstations. Note that the VAXstation workstations use a white background by default.

Index	Color	Red Intensity	Green Intensity	Blue Intensity
0	White	1.0	1.0	1.0
1	Black	0.0	0.0	0.0

If you request a shade that is above 50% on the gray scale, the VAXstatio handler uses the black representation; otherwise, it uses the white representation.

The following table lists the default red, green, and blue intensity associated with each color index value. The number of default colors is anywhere from 2 to 248, depending on the GPX bit planes. (For more information about GPX bit planes, see the VAX GKS documentation.) Be aware that these values can be redefined with the SET COLOR MIX statement, and that a change to the color intensity associated with red, green, or blue alters the color associated with the color index. The first eight default colors and the equivalent intensity values are as follows:

Index	Color	Red Intensity	Green Intensity	Blue Intensity	Gray Level
0	White	1.0	1.0	1.0	1.0
1	Black	0.0	0.0	0.0	0.0
2	Red	1.0	0.0	0.0	0.30
3	Green	0.0	1.0	0.0	0.59
4	Blue	0.0	0.0	1.0	0.11
5	Cyan	0.0	1.0	1.0	0.60
6	Magenta	1.0	0.0	1.0	0.41
7	Yellow	1.0	1.0	0.0	0.89

Color indices 8 through 32 are random colors defined so that you can use any two colors side by side without having difficulty distinguishing between colors. Indices 33 through 248 are defined such that first the blue intensity value is incremented, then the green intensity, and finally the red intensity.

.3 Color Intensities

The following chart illustrates the possible colors that you can display by supplying the indicated intensity values with the SET COLOR MIX statement.

Figure B–5 presents a chart of 64 colors and their corresponding red, green, and blue intensity values. If you are working with a color VT125, a VT241, or a VAXstation II/GPX, you can use this color chart as a guide when using the SET COLOR MIX statement. The colors presented are the 64 colors supported by the VT125 and the VT241.

You should use this color chart as a guide. You should not expect your monitor to display the colors exactly as shown. Colors can vary from monitor to monitor depending on the following factors:

- The current background color (affects lighter shades)
- The current brightness and contrast control settings
- The available room light
- The proximity of the primitive to other colors on the display

Figure B–5: Intensity Values for VT125 and VT240 Terminal and VAXstations

Red	Green	Blue	
0.0000	0.0000	0.0000	■
0.1400	0.1400	1.0000	■
1.0000	1.0000	1.0000	□
1.0000	0.6133	0.4200	▨
1.0000	0.5700	0.1400	▨
0.8538	0.6646	0.2862	▨
0.8400	0.5600	0.0000	▨
1.0000	0.1400	0.1400	■
0.8400	0.0000	0.0000	■
0.8400	0.0000	0.5600	■

(Continued on next pag

Figure B–5 (Cont.): Intensity Values for VT125 and VT240 Terminals and VAXstations

Red	Green	Blue	
1.0000	0.1400	0.5700	▨
0.8538	0.2862	0.6646	■
1.0000	0.1400	1.0000	■
1.0000	0.4200	0.6133	▨
0.0000	0.0000	0.5600	■
0.6133	0.4200	1.0000	▨
0.4200	1.0000	1.0000	▨
0.5600	0.0000	0.8400	■
0.7765	0.9235	0.9235	▨
0.2862	0.6646	0.8538	▨

(Continued on next page)

Figure B–5 (Cont.): Intensity Values for VT125 and VT240 Terminals and VAXstations

Red	Green	Blue	
0.2862	0.2862	0.8538	▓
0.4200	0.6133	1.0000	░
0.1400	1.0000	1.0000	▒
0.5700	0.1400	1.0000	▒
0.0000	0.5600	0.8400	▓
0.0000	0.0000	0.8400	█
0.1400	0.5700	1.0000	█
0.0000	0.8400	0.5600	█
0.0000	0.5600	0.0000	█
0.1400	1.0000	0.5700	▓

(Continued on next page)

Figure B–5 (Cont.): Intensity Values for VT125 and VT240 Terminals and VAXstations

Red	Green	Blue	
0.5700	1.0000	0.1400	■
0.1400	1.0000	0.1400	■
0.0000	0.8400	0.0000	■
0.2862	0.8538	0.2862	■
0.5600	0.8400	0.0000	■
1.0000	1.0000	0.7000	▨
0.6646	0.8538	0.2862	▨
0.9235	0.9235	0.7765	□
0.7119	0.7119	0.4281	▨
0.5600	0.5600	0.0000	■

(Continued on next page)

Figure B–5 (Cont.): Intensity Values for VT125 and VT240 Terminals and VAXstations

Red	Green	Blue	
1.0000	1.0000	0.1400	▨
0.6700	0.6700	0.6700	▨
0.8521	0.5679	0.5679	▨
0.3300	0.3300	0.3300	▨
0.5600	0.0000	0.0000	■
0.8538	0.2862	0.2862	▨
0.6258	0.2142	0.2142	▨
0.5600	0.0000	0.5600	■
0.7119	0.4281	0.7119	▨
1.0000	0.4200	1.0000	▨

(Continued on next page

Figure B–5 (Cont.): Intensity Values for VT125 and VT240 Terminals and VAXstations

Red	Green	Blue	
1.0000	0.7000	1.0000	▨
0.6646	0.2862	0.8538	▨
0.9235	0.7765	0.9235	▨
0.2142	0.2142	0.6258	▨
0.7000	1.0000	1.0000	▨
0.5679	0.5679	0.8521	▨
0.0000	0.5600	0.5600	■
0.4281	0.7119	0.7119	▨
0.2862	0.8538	0.6646	▨
0.5679	0.8521	0.5679	▨

(Continued on next page)

Figure B–5 (Cont.): Intensity Values for VT125 and VT240 Terminals and VAXstations

Red	Green	Blue	
0.2142	0.6258	0.2142	■
0.4200	1.0000	0.6133	▨
0.6133	1.0000	0.4200	▨
1.0000	1.0000	0.4200	░

INDEX

Programming with
VAX BASIC Graphics
AI–HY14A–TE

READER'S COMMENTS

Note: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent:

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code or Country ______

— — — — Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

— — — — Do Not Tear - Fold Here —

Cut Along Dotted Line