RGL/11 Programmer's Reference Manual

AA–M837A–TC

August, 1982

This manual describes the RGL/11 (ReGIS Graphics Library) package of graphic subroutines for RT–11 and RSX–11M operating systems.

This is a new document.

OPERATING SYSTEM:	RT–11, V4.0 RSX–11M, V4.0
SOFTWARE:	RGL/11, V1.1
On RT–11:	FORTRAN IV, V2.5
On RSX–11M:	FORTRAN 77, V4.1

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region	**Central Region**	**Western Region**
Digital Equipment Corporation PO Box CS2008 Nashua, New Hampshire 03061 Telephone:(603)884–6660	Digital Equipment Corporation Accessories and Supplies Center 1050 East Remington Road Schaumburg, Illinois 60195 Telephone:(312)640–5612	Digital Equipment Corporation Accessories and Supplies Center 632 Caribbean Drive Sunnyvale, California 94086 Telephone:(408)734–4915

digital equipment corporation • marlboro, massachusetts

First Printing, August 1982

Contents

Figures

Preface

RGL/11 is a package of picture–drawing and data–plotting subroutines that run on RT–11 and RSX–11M operating systems. RGL/11 requires a VT125 graphics terminal.

"RGL/11" is a mnemonic for the *R*eGIS *G*raphics *L*ibrary. "ReGIS" is a mnemonic for the "*Re*mote *G*raphics *I*nstruction *S*et." The VT125 terminal uses this instruction set to generate graphic images on the terminal screen.

RGL/11 is a subset of the full VAX–11 RGL library. All subroutine names, argument names, and the order of the arguments are upward–compatible with the VAX–11 RGL package.

In order to program graphic applications, you do not need to know graphic hardware or ReGIS commands. You need only include the appropriate RGL/11 subroutine(s) in your FORTRAN programs.

Audience

This manual assumes you know your operating system — RT–11 or RSX–11M — FORTRAN programming, and how to set up your VT125 terminal.

Associated Documentation

For information on the VT125 terminal, refer to:

- *VT125 User's Guide* (EK–VT125–UG)

For information about installing RGL/11 on an RSX–11M operating system and building executable programs, refer to:

- *RGL/11 Installation Guide* (AA–M838A–TC)

- *RSX–11M/M–PLUS MCR Operations Manual* (AA–H263A–TC)
- *RSX–11M/FEP and RSX–11M/FRP Installation and User's Guide* (AA–J303C–TC)

For information about installing RGL/11 on an RT–11 operating system and building executable programs, refer to:

- *RGL/11 Installation Guide* (AA–M838A–TC)
- *RT–11/FEP and RT–11/FRP Installation and User's Guide* (AA–M079B–TC)
- *RT–11 System User's Guide* (AA–5279B–TC)

For information on FORTRAN 77 (for RSX systems), refer to:

- *PDP–11 FORTRAN 77 Language Reference Manual* (AA–1855E–TC)
- *PDP–11 FORTRAN 77 User's Guide* (AA–1884D–TC)
- *PDP–11 FORTRAN 77 Installation Guide/Release Notes* (AA–K503B–TC)

For information on FORTRAN IV (for RT–11 systems), refer to:

- *PDP–11 FORTRAN Language Reference Manual* (AA–1855D–TC)
- *RT–11/RSTS/E FORTRAN IV User's Guide* (AA–5749B–TC)
- *RT–11 FORTRAN IV Installation Guide* (AA–5240D–TC)

Manual Organization

Chapter 1 introduces the RGL/11 subroutine package and the VT125 terminal.

Chapter 2 introduces picture–drawing concepts and strategy.

Chapter 3 introduces data–plotting concepts and strategy.

Chapter 4 describes creating programs that have RGL/11 subroutine calls in them.

Chapter 5 describes each RGL/11 subroutine, giving:

- its purpose
- its syntax
- descriptions of its arguments
- an example of a call
- related subroutines

- restrictions on its effect
- error messages it can produce
- a sample program fragment that uses it

Appendix A presents a one–sentence summary of the subroutines listed alphabetically.

Appendix B lists RGL/11 error messages and error codes, the subroutine or subroutines that can cause the error, and a description of the circumstances that caused the error.

Glossary contains the graphics vocabulary used in the manual.

Manual Conventions

Braces { } indicate that the enclosed arguments are optional. For example, the following statement means that all arguments are optional:

CALL MARKER ({number}, {x}, {y})

Although the arguments are optional, the parentheses and commas must be supplied.

RET indicates that you should press the RETURN key.

DEL indicates that you should press the DELETE key.

filespec In RT–11, the format is:

```
dev:filnam.ext
```

For more information about file specifications, refer to the *RT–11 System User's Guide.*

In RSX–11M, the format is:

```
dev:[group,member]filename.extension;version
```

For more information about file specifications, refer to the *RSX–11M/M–PLUS MCR Operations Manual.*

To reference a file extension, you must supply one.

LUNs Logical Unit Numbers. RGL/11 reserves two LUNs:

LUN	Reserved for
1	the error–handling subroutine
5	the terminal

RGL/11 establishes certain LUNS as follows: LUN 2 as the device that GLOAD and LCHRST read from, and LUN 3 for the device GSAVE writes to.

The LUN default setting enables six LUNs to be open at the same time. To enable more LUNs in an RT–11 system, you must use the /UNITS: option at compile time (for more information refer to the *RT–11/RSTS/E FORTRAN IV User's Guide*). To create more LUNs in an RSX–11M system, you must use the /UNITS= option at task–build time (for more information, refer to the *RSX–11M/M–PLUS Task Builder Manual*).

Chapter 1 Introduction

This chapter gives an overview of the RGL/11 subroutines and the VT125 graphics terminal.

1.1 Introduction to the RGL/11 Package

"RGL/11" (*R*eGIS *G*raphics *L*ibrary for RT–11 and RSX–11M operating systems) is the name of a package of graphic subroutines that can draw pictures and plot data on the VT125 terminal. When you want to draw images or graphs, you write an application program that calls the appropriate RGL/11 subroutines. The subroutines then generate the ReGIS commands that create and display the graphic objects or graphs on the screen.

"*Graphic objects*" refer to both picture objects and graphic text. A *picture object* is any graphic object except graphic text, such as a line, polygon, circle, or arc. Nine different line patterns are available for drawing the picture objects. If you want to shade a picture object, you can use a shading option and select the shading pattern and color you want. You can select one of four gray shades or, if you have the optional color monitor, one of four colors.

The RGL/11 subroutines are treated individually in Chapter 5 and are summarized in Appendix A. Using the subroutines, you can:

- plot data by calling just one subroutine, PDATA.
- plot data from stored arrays or interactively plot individual data points.
- create linear and logarithmic grids.
- create multi–scaled graphs that have two y–axes scaled and one x–axis scaled.

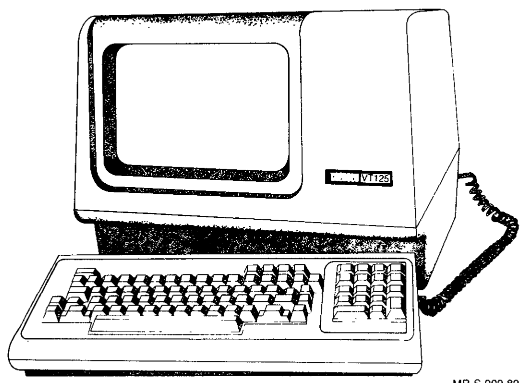
- label the cells of an axis with either number labels or text labels, and label the axis with a title.
- find the coordinates of points on a graph interactively.
- add points to an existing graph.
- draw regular polygons, circles, and arcs anywhere on the screen having the size, the location, and, for polygons, the number of sides you specify. You can select the type of line (such as a solid line, or a line of dots and dashes) and whether or not an area is shaded. If the area is shaded, you have a choice of shade pattern and color.
- produce graphic images on the screen from files stored on any storage medium your operating system supports.
- interactively get the coordinates of any location on the screen.
- draw pictures relative to the current graphic cursor location or to the origin location.
- save graphic files on mass–storage devices and restore them.
- control whether or not RGL/11 error messages are displayed on your screen.
- control whether angle arguments are interpreted as radians (the default) or as degrees.
- specify the coordinate system of your picture.
- write graphic text which you can easily place anywhere on the screen (RGL/11 handles the necessary FORMAT instructions), enlarge the graphic text to up to 16 times its normal size, and write it in any one of four colors.
- write in the Greek character set as well as in the English (default) set.
- write in different "modes" so that you can create effects of overwriting, reversing color, and so forth.
- erase the screen completely or selectively.
- use an optional LA34–VA printer to print copies of images that are on the graphic screen.

1.2 Introduction to the VT125 Graphics Terminal

The VT125 terminal is a VT100 terminal with additional firmware that enables the terminal to display graphic images as well as text. The terminal's black–and–white monitor can display up to four shades of gray. If you want color displays, you can connect a color monitor to the VT125. You also have the option of attaching a LA34–VA printer to the VT125 for hard copies of graphs and other graphic images.

RGL/11 uses two modes of the VT125 terminal: text mode and graphic mode. In text mode, it operates as a standard VT100. In graphic mode, it can process instructions to draw lines, circles, polygons, and arcs as well as text. When the terminal is first turned on, it is in text mode.

Figure 1–1: The VT125 Graphics Terminal



1.2.1 The VT125 Text Mode

Text mode is the initial default state of the VT125 terminal. In text mode, the terminal operates like a regular ASCII VT100 terminal. It displays a *text mode cursor*, the regular VT100 cursor, that indicates where text will be displayed. The software that controls the terminal specifies where a message is displayed by row and column. There are 24 rows and 80 columns. After the message is written, the cursor is at the end of the message.

1.2.2 The VT125 Graphic Mode

NOTE

> When you plan to use graphic programs, set the terminal driver so that the operating system will not insert carriage return/line feeds into lines whose width is longer than the terminal's line width (NOCRLF for RT, NOWRAP for RSX). For the commands that set the terminal driver, refer to Chapter 4, Program Development.

When the terminal is in graphic mode, the *graphic cursor* is displayed. The cursor is a blinking, diamond-shaped object. Like the text cursor, it marks where the graphic object will be displayed. Often when graphic programs are executing, the cursor moves so fast it is invisible; however, its location is always stored internally. The "current graphic cursor location" refers to this stored location.

In text mode, the text screen is addressed by row and column. In graphic mode, there are many more addressable locations because the graphic screen is addressed by individual *pixels*. A pixel is a picture element, the smallest displayable unit on the screen. (For addressing the screen in graphic mode, see Section 2.2, Defining Coordinate Systems.) If, however, you want to write text in graphic mode and address the graphic screen by row and column coordinates, not pixels, you can do so. The subroutine LINETX addresses the graphic screen by row and column.

In graphic mode, the terminal interprets special graphic commands sent to it by the RGL/11 software to plot linear and logarithmic graphs, draw graphic objects, write graphic text in sizes that range from standard size to 16 times larger than standard, and perform other features of graphics such as shading, line pattern variations, and so forth. These graphic commands are in the RGL/11 package, described in Section 1.1.

1.2.3 VT125 Grays and Color

The VT125 is equipped with a black–and–white monitor that can display four shades of gray. If you attach a color monitor, the terminal will display in black and white on its screen and simultaneously display colors on the color monitor's screen. The gray shades, GRAY0 through GRAY3, on the color monitor are displayed as dark, blue, red, and green, respectively.

One of the shades, GRAY0, acts as the default *screen color*. The other three colors are available to draw and shade graphic objects; the color selected for drawing or shading is the *drawing color.*

1.2.4 Setting VT125 Characteristics

You can alter some of the terminal's characteristics either before using the terminal (in a login command file, for example) or while the terminal is in use. See the *VT125 User's Guide* for more information.

1.2.5 Controlling Data Being Transmitted to the Screen

When the VT125 displays a file, it displays the data continuously until it reaches the end of the file. If you want to halt the display of data, you have two choices. You can halt the display and resume it with no loss of data, or you can halt it and "discard" the data being transmitted.

To halt the display and discard data, type CTRL/O (hold down the <CTRL> key while you type the letter "O"). To cause the display to resume, type CTRL/O again. The display resumes, but all data transmitted to the terminal since you first typed CTRL/O is lost.

To halt the display without losing data, press the <NO SCROLL> key which is located at the bottom left of the main keyboard. To cause the display to resume, press <NO SCROLL> again. The display resumes where it left off when you first pressed <NO SCROLL>. No data has been lost.

For this <NO SCROLL> mechanism to be available to you, you must set the terminal characteristic AUTO XOFF/XON to ON. You set it by using the terminal's SET UP keys as follows:

1. Press the SET–UP key at the upper left of the keyboard to put the terminal in SET–UP A mode.
2. Press the 5 key on the main keyboard to put the terminal in SET–UP B mode.
3. The terminal displays five groups of binary values. Look at the fourth entry in the second group of values. If that value is one, then AUTO XOFF/XON is correctly set; press the SET–UP key again to return the terminal to normal operation and ignore steps 4, 5, 6, and 7 below. If that value is zero, then AUTO XOFF/XON is not set, and you should perform steps 4, 5, 6, and 7.
4. Press the right–arrow (→) key, which is located at the upper right corner of the main keyboard, until the cursor is positioned over the fourth entry in the second group of displayed values.
5. When the cursor is correctly positioned over the fourth entry, press the 6 key on the main keyboard to change the value from zero to one.
6. Save the current SET–UP features by typing SHFT/S. The terminal clears the screen, displays WAIT, and returns to SET–UP A mode.
7. Press the SET–UP key again to return the terminal to normal operation.

This procedure is explained in the *VT125 User's Guide* referenced in Section 1.2.4.

For information on how to erase the screen completely, see Section 2.1.1.

Chapter 2
Picture Drawing

This chapter describes each aspect of picture drawing, and gives sample programs. It describes how to draw graphic objects, generate different line patterns, set shading, set writing modes, set the drawing color, and so forth. The subroutines that perform these tasks are described in detail in Chapter 5, and information on creating and running graphic programs is given in Chapter 4.

If you are interested only in data plotting, you do not need to read this chapter. Instead, refer to Chapter 3, Data Plotting.

The sections of this chapter are:

This list does not include all the picture–drawing subroutines. Those that are not included are subroutines that copy the graphic images to hard copy (CPYSCR), define what unit of measure the angle argument will use (SRADNS and SDGREE), and determine whether or not RGL/11 error messages will be displayed on the screen (SDEBUG and SNDBUG).

NOTE

The sample programs in this chapter are also files on your distribution volume. These demonstration files use the file extension .DEM. This extension makes it easy to get a directory listing of all demo programs. When you use the RGLLNK indirect command file to run them, you need to supply both the file name and the .DEM extension. (For RGLLNK on RT–11 systems, see Section 4.3.2.2); on RSX–11M systems, see Section 4.4.2.2).

The file names, but not extensions, are given in the following sections.

2.1 General Strategy for Drawing Graphic Objects

When you write a graphic program, you must first initialize the terminal so that all its attributes (such as character size and graphic cursor location) are at a known value. You do this by calling the INITGR (*Init*ialize *Gr*aphics) subroutine. Next you make sure the screen is clear of graphic or text images with calls to CLRSCR (*Cl*ear *Scr*een) and CLRTXT (*Cl*ear *T*ext). Third, you usually establish the world coordinate system you want with a SWINDO (*S*et *Windo*w) call.

Next you move the graphic cursor to the location where you want to start drawing the object, specifying the location in the world coordinate system you established. Then you use the subroutine to draw the object. After the object is drawn, the graphic cursor may be at the location where you want to draw the next graphic object. If so, you can call the subroutine that draws that object; if not, you call the MOVE subroutine to move the graphic cursor. (Three subroutines have arguments for coordinates of a starting location, and therefore they do not need to be preceded by a call to MOVE: BOX, LINETX, and MARKER.)

The following example, named OBJECT on your distribution volume, draws several graphic objects and graphic text. It also enables the locator cursor so that you can read locator coordinates and/or experiment by adding objects to the picture (see Section 2.1.2). Figure 2–1 shows the picture created by this example.

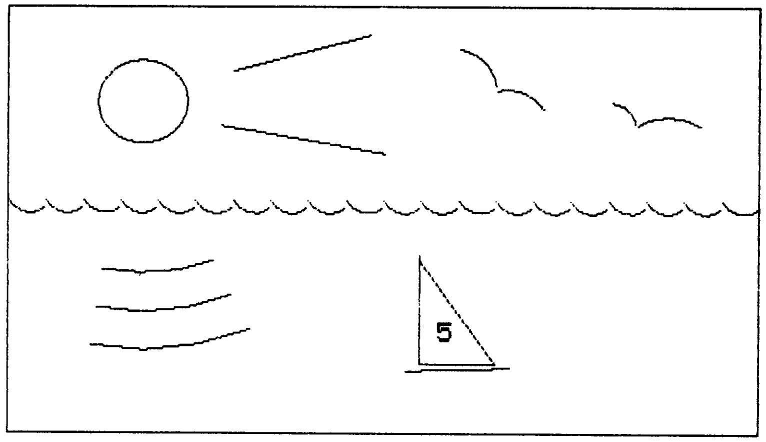
```
C    OBJECT
C
           CALL INITGR (5)
           CALL CLRSCR
           CALL CLRTXT
           CALL SWINDO (0.,0.,1000.,625.)
C    This call to BOX creates the picture's frame.
           CALL BOX (0.,0.,1000.,625.)
C
C    These calls create the sun, arcs in water, and rays in sky.
           CALL MOVE (180.,478.)
           CALL CIRCLE (60.)
           CALL SDGREE
```

```
          CALL ARC (-35.,275.,250.)
          CALL ARC (-35.,300.,200.)
          CALL ARC (-35.,325.,150.)
C
          CALL MOVE (300.,523.)
          CALL LINE (483.,575.)
          CALL MOVE (285.,444.)
          CALL LINE (502.,404.)
C
C    This loop creates the waves.
          X=30.
          XX=50.
          DO 10 I=1,20
          CALL MOVE (X,350.)
          CALL ARC (-100.,XX,325.)
          X=X+50.
          XX=XX+50.
 10       CONTINUE
C
C    These arcs form the seagulls.
          CALL MOVE (590.,496.)
          CALL ARC (-75.,600.,556.)
          CALL MOVE (652.,496.)
          CALL ARCC (-75.,665.,440.)
C
          CALL MOVE (835.,444.)
          CALL ARCC (80.,800.,444.)
          CALL MOVE (838.,444.)
          CALL ARCC (-70.,880.,384.)

C    These calls create the boat.
          CALL MOVE (550.,260.)
          CALL LINE (550.,100.)
          CALL LINE (650.,100.)
          CALL LINE (550.,250.)
          CALL MOVE (530.,90.)
          CALL LINE (670.,94.)
          CALL STXSIZ (2,)
          CALL MOVE (570.,165.)
          CALL TEXT ('5')

C    This section enables the locator cursor.  When you
C    type any key but the SHIFT key, ESCAPE key, or RETURN
C    key, the location of the locator cursor is displayed
C    on the screen.  When you type RETURN, the program
C    terminates.  Move the cursor by means of the arrow
C    keys at the upper right of the keyboard.
C
 50       CALL LOCATE (X, Y, KEY)
          IF (KEY.EQ."177) GO TO 50
          TYPE 100, X, Y, KEY
 100      FORMAT (' THE LOCATION IS ',2F8.2, ' AND KEY IS ',A1)
          IF (KEY.NE."015) GO TO 50
C
          END
```

Figure 2–1: Picture Drawn by Program OBJECT



2.1.1 Clearing the Screen

To clear the image and text from the screen we create and run the CLEAR program. The text of this file is:

```
CALL INITGR(5)
CALL CLRTXT
CALL CLRSCR
END
```

2.1.2 Modifying a Graphic Program

Now we decide to add another sailboat to the picture, to change the line pattern of the picture frame, and to make the sun a solid.

Good programming practice suggests that we copy the original file and experiment on the copy. Therefore we copy OBJECT.DEM to a file we name OBJ2.DEM.

To add a sailboat to the picture, we use the locator cursor to get coordinates for three points of the sail and two points for the prow and stern of the boat. Then we add the necessary commands to OBJ2.DEM to make the changes we have decided on.

First, we compile, link, and run OBJ2.DEM. When the picture appears, we decide where we want the sailboat to go, and use the locator cursor to get the coordinates we need. We choose these coordinates:

X–coordinates	Y–coordinates
743.	251. (apex of sail)
743.	160. (lower left corner)
808.	160. (lower right corner)
738.	151. (prow)
812.	154. (stern)

You may want to use these coordinates or you may want to get your own set of coordinates. When you are finished, use the RETURN key to exit the program, and then clear the screen.

Next we edit OBJ2.DEM. After the line:

```
CALL TEXT ('5')
```

we insert the following code:

```
C   This code adds the new sail.
          CALL MOVE (743.,251.)
          CALL LINE (743.,160.)
          CALL LINE (808.,160.)
          CALL LINE (740.,251.)
C   This code adds the hull.
          CALL MOVE (728.,151.)
          CALL LINE (812.,154.)
```

Now we change the line pattern of the picture frame. Before the line:

```
CALL BOX (0.,0.,1000.,625.)
```

we insert this line:

```
CALL SLNPAT (7,5)
```

This line uses line pattern 7 (see other possible line patterns in Chapter 5, SLNPAT) and multiplies the size of the basic pattern 5 times.

To return the line pattern to a solid line, the default pattern, we insert after the CALL BOX line:

```
CALL SLNPAT (,)
```

Next we change the sun to a solid circle by inserting a new call before this call:

```
CALL CIRCLE (60.)
```

We want the sun to be shaded with the "%" character, so we insert this command:

```
CALL SSHADE (,'%')
```

To keep the rest of the picture objects unshaded, we return to the default (no–shade) state by inserting this call after the CALL CIRCLE line:

```
CALL SNSHAD
```

After the edits are complete, we compile, link, and run OBJ2.

2.2 Defining Coordinate Systems

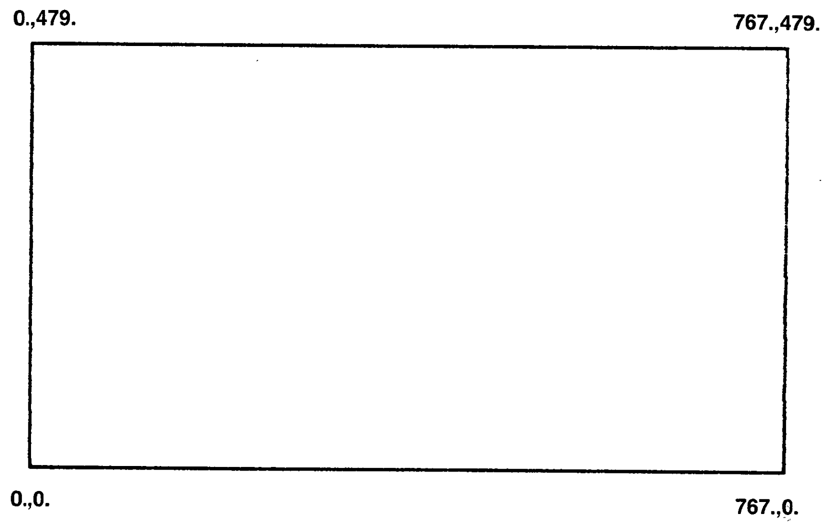
NOTE

The programs that created the figures in this section are not on your distribution volume.

Coordinate systems are the x–axis and y–axis values that define the space on the screen. RGL/11 subroutines work with two types of coordinate systems: the *physical screen coordinate system* and the *world coordinate system.*

The *physical screen coordinates* are the coordinates the terminal hardware understands. For the VT125, they range from 0 to 767 in the x–direction and 0 to 479 in the y–direction. (The terminal has its origin location in the upper left corner of the screen, but RGL/11 sets it to the lower left because that is where most people expect it to be.) Figure 2–2 shows the physical screen coordinate range that RGL/11 uses.

Figure 2–2: VT125 Screen Coordinates



However, you are not restricted to the screen coordinates. The images you work with may use ranges other than 0 to 767 and 0 to 479, you may want the entire screen to display just a segment of the images, or you may want to shrink the image to a small portion of the screen. The RGL/11 subroutine SWINDO gives you a way to redefine the coordinates you use to address the screen. The coordinates you establish using SWINDO are called *world coordinates.* The SWINDO format is:

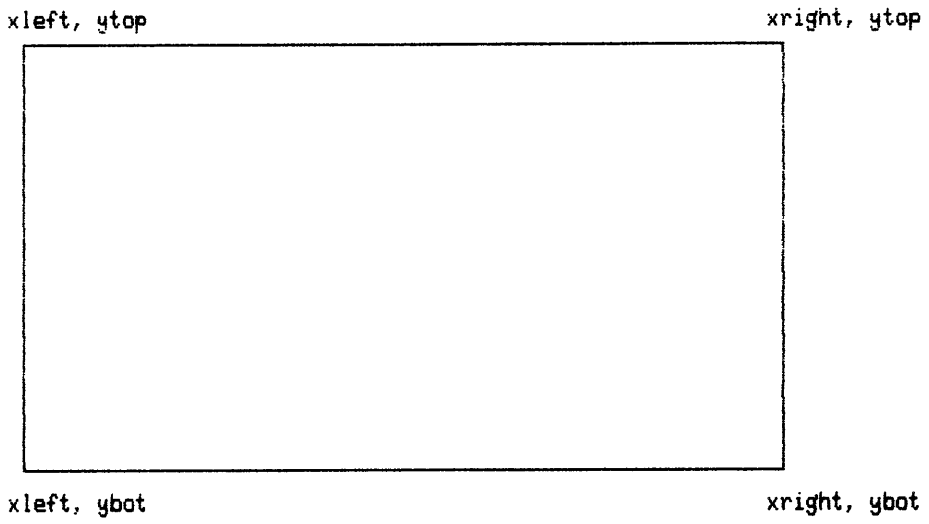
SWINDO (xleft, ybot, xright, ytop)

where:

- xleft is the x–coordinate at the left edge of the picture
- ybot is the y–coordinate of the bottom edge

- xright is the x–coordinate of the right edge
- ytop is the y–coordinate of the top edge

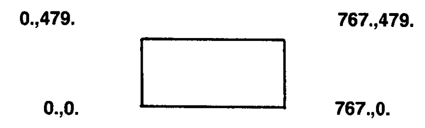
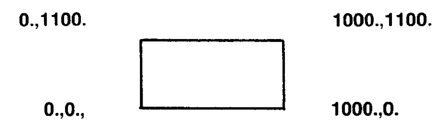
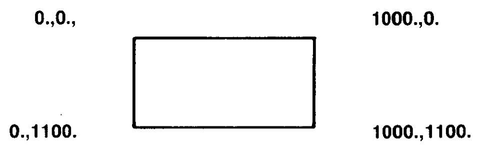
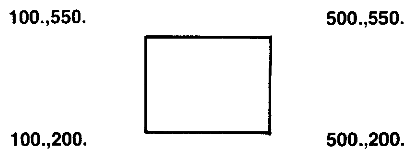
Figure 2–3: SWINDO Arguments



When you establish a world coordinate system, you usually want to maintain the aspect ratio of the screen; the aspect ratio is the ratio of the horizontal axis to the vertical. The aspect ratio is 768 to 480, or 8 to 5. If the aspect ratio is different from 768 to 480, then picture objects will be distorted. For example, circles look more like ellipses if the aspect ratio is different.

Some examples of world coordinate systems follow.

Table 2–1: Examples of World Coordinate Systems

xleft	ybot	xright	ytop	Example
0.0	0.0	767.0	479.0	
0.0	0.0	1000.0	1100.0	
0.0	1100.0	1000.0	0.0	
100.0	200.0	500.0	550.0	

MR-S-1598-81

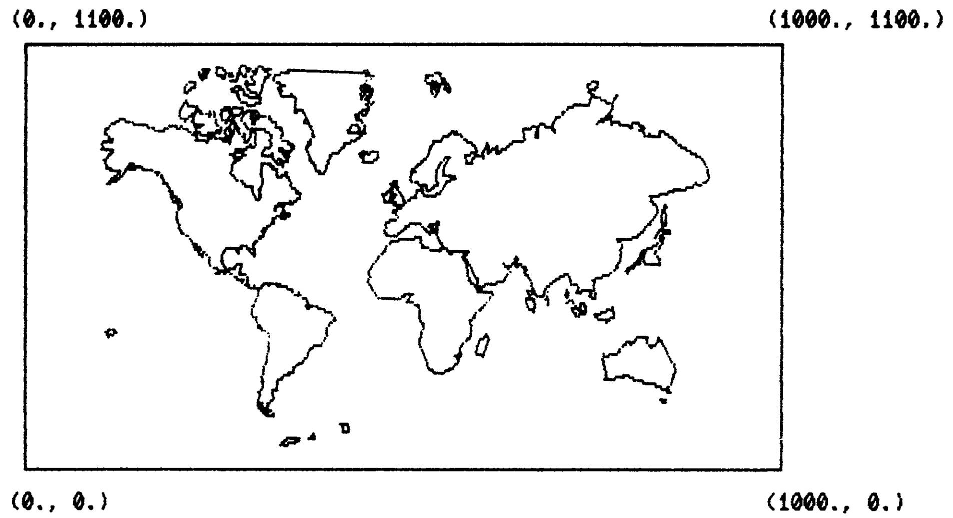
1. `SWINDO (0.0, 0.0, 767.0, 479.0)`

   This SWINDO call reestablishes the *screen coordinate system.* Notice that the arguments are in floating–point notation.

2. `SWINDO (0.0, 0.0, 1000.0, 1100.0)`

   If you wanted to draw a map that was measured in meters, you would want your *world coordinate system* to be in meters. This call gives the map a scale of 1100 meters high by 1000 wide.

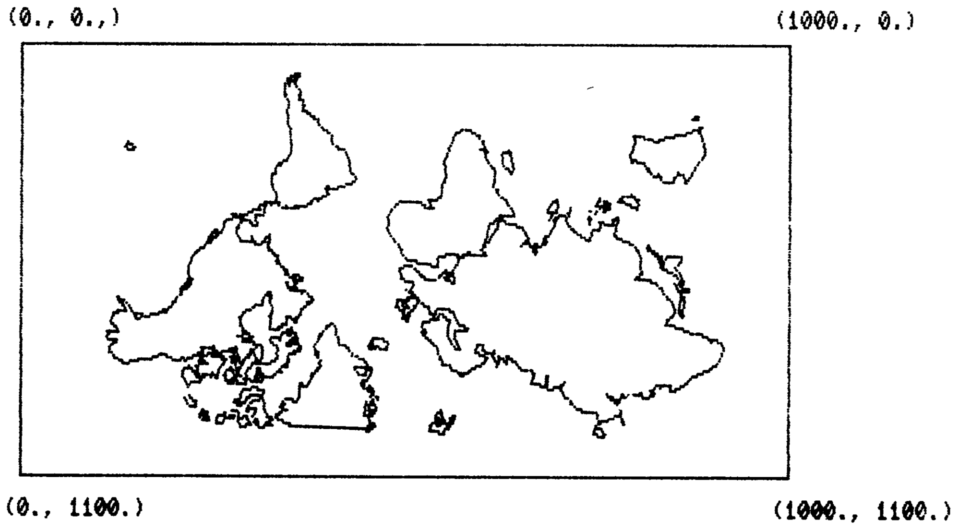
Figure 2–4: SWINDO Example 1



3. `SWINDO (0.0, 1100.0, 1000.0, 0.0)`

   If you wanted to change the *origin location* of your coordinate system, you would define the arguments accordingly. A common alternate origin location is the top left, as in this call; this location is the origin location of some digitizers. When a digitizer supplies your map's coordinates and its origin location is in the upper left corner of the digitizer pad, you would match its orientiation by defining the upper left of your coordinate system as the origin location. You do this by defining the ytop and xleft arguments as 0.0 and the ybot and xright arguments as some larger number. The picture below was created by making a call to SWINDO to set the origin to the upper left and then redrawing the map in Figure 2–4.

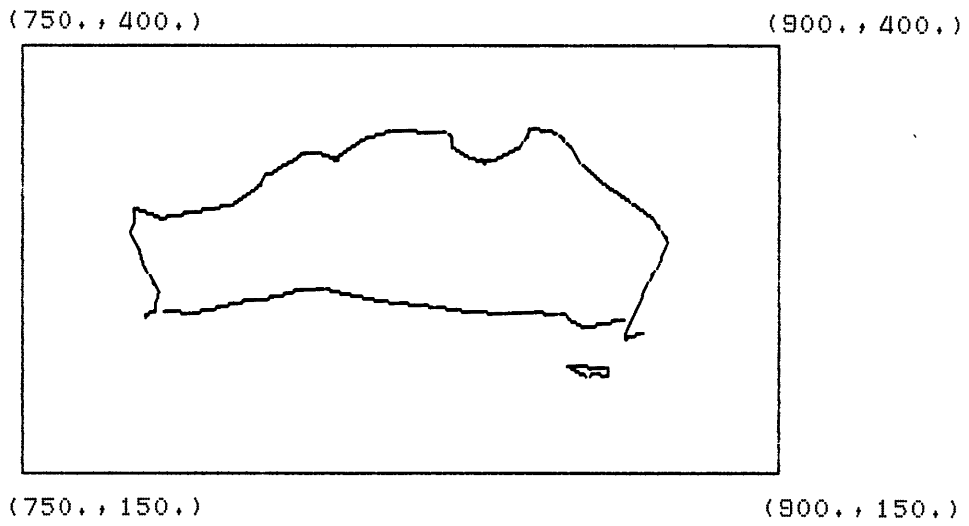
Figure 2–5: SWINDO Example 2



4. `SWINDO (750.0, 150.0, 900.0, 400.0)`

You can also use SWINDO to display only *a portion of your entire picture*. For example, this call displays only the part of the map that is between 750 and 900 units in the x–direction and between 150 and 400 units in the y–direction. The picture below was drawn by a call to SWINDO with the smaller coordinate range followed by the call to redraw the picture again. All points not within the window are clipped by the software.

Figure 2–6: SWINDO Example 3



2.3 Drawing Graphic Objects from Data Arrays

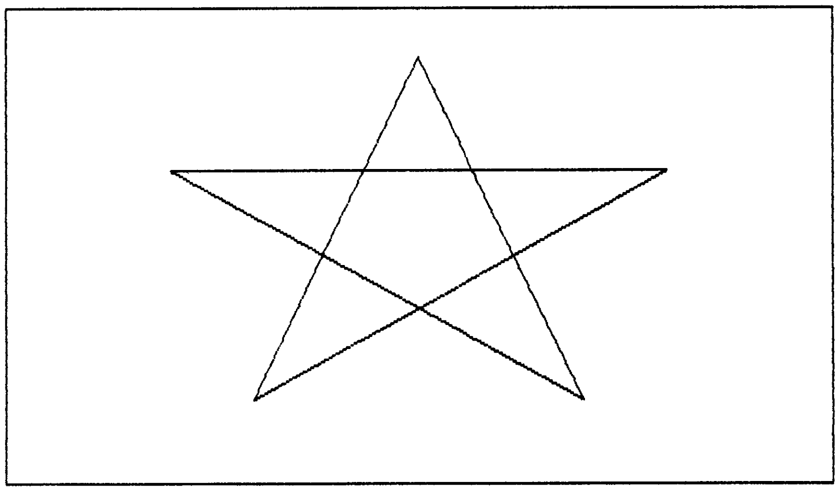
You can draw graphic objects from data that is stored in arrays by calling either the POLYLN (*Poly Line*) or RELPLN (*Rel*ative *P*oly *Line*) subroutine. The POLYLN subroutine uses absolute locations, the x– and y–coordinates of the endpoints of the lines. The RELPLN subroutine uses relative locations, the x– and y–distances from the current graphic cursor location. The arguments for both subroutines are the number of lines to draw and the names of the arrays containing the x and y data points.

The following program, named STAR on your distribution volume, uses the POLYLN subroutine.

```
C  STAR
C  This program uses the POLYLN subroutine to draw a star.
C
       DIMENSION XSTAR (5), YSTAR (5)
       DATA XSTAR /500.0,700.0,200.0,800.0,300.0/
       DATA YSTAR /560.0,110.0,410.0,410.0,110.0/
```

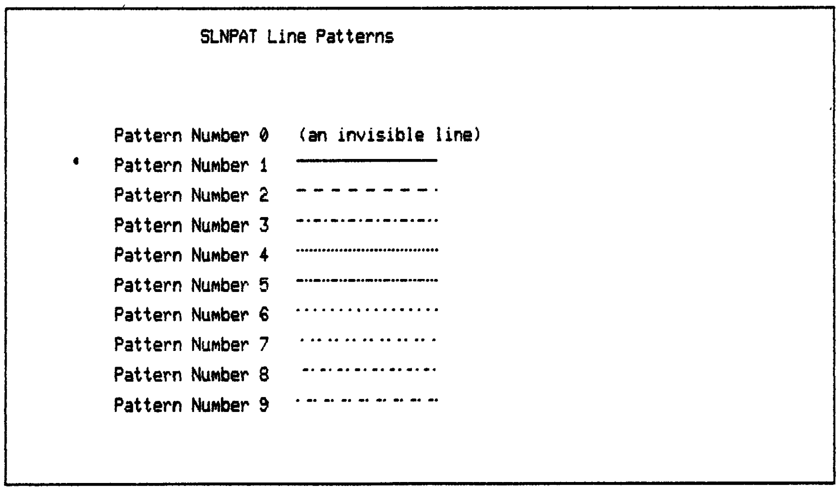
```
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
        CALL BOX (0.,0.,1000.,625.)
C
C   Move to the starting location.
C
        CALL MOVE (300.0, 110.0)
C
C   Draw the star.
C
        CALL POLYLN (5, XSTAR, YSTAR)
        END
```

Figure 2–7: POLYLN Example in the STAR Program



2.4 Changing the Line Pattern

When you draw graphic objects, you can change the pattern of the line by calling a subroutine named SLNPAT (*S*et *L*i*n*e *Pat*tern). The line patterns you can select are:



```
SLNPAT Line Patterns

Pattern Number 0   (an invisible line)
Pattern Number 1
Pattern Number 2
Pattern Number 3
Pattern Number 4
Pattern Number 5
Pattern Number 6
Pattern Number 7
Pattern Number 8
Pattern Number 9
```

Each line pattern has a number assigned to it. You call SLNPAT as follows:

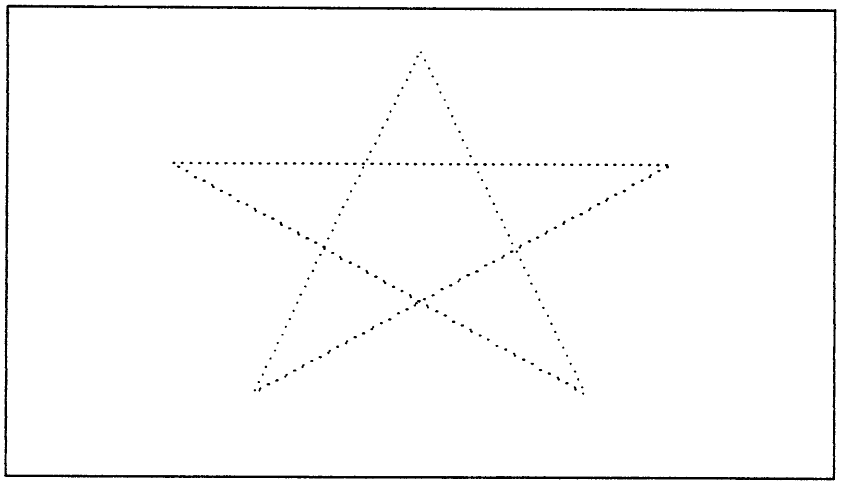
CALL SLNPAT ({number}, {mult})

where "number" specifies the line pattern, and "mult" specifies the pattern multiplier. The new line pattern stays in effect until changed by another SLNPAT call or by an INITGR call. The multiplier multiplies the size of each component of the pattern. For example, line pattern 3 uses a long dash, a space, and a short dash. Each of those components is multiplied by the pattern multiplier.

The following program, named LINPAT on your distribution volume, uses the SLNPAT subroutine.

```
C   LINPAT
C   The following program sets the line pattern to a dashed line
C   and then draws a star by calling the subroutine POLYLN.
C
C   Fill the x- and y-arrays with the endpoints of the lines.
      DIMENSION XSTAR (5), YSTAR (5)
      DATA XSTAR /500.0, 700.0, 200.0, 800.0, 300.0/
      DATA YSTAR /560.0, 110.0, 410.0, 410.0, 110.0/
C
      CALL INITGR (5)
      CALL CLRSCR
      CALL CLRTXT
      CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
      CALL BOX (0.,0.,1000.,625.)
C
C   Set the line pattern to be pattern 6 with a multiplier of 2.
C
      CALL SLNPAT (6, 2)
C
C   Move to the starting location.
C
      CALL MOVE (300.0, 110.0)
C
C   Draw the star.
C
      CALL POLYLN (5, XSTAR, YSTAR)
      END
```

Figure 2–8: SLNPAT Example in the LINPAT Program



2.5 Shading Picture Objects

To "shade" picture objects means to affect every pixel from every point in the picture object to a *shade line* you specify, using the shade pattern and color of your choice. The SSHADE (*S*et *Shade*) subroutine enables you to shade picture objects with the line patterns listed in the SLNPAT subroutine or with the characters of the character set you have enabled.

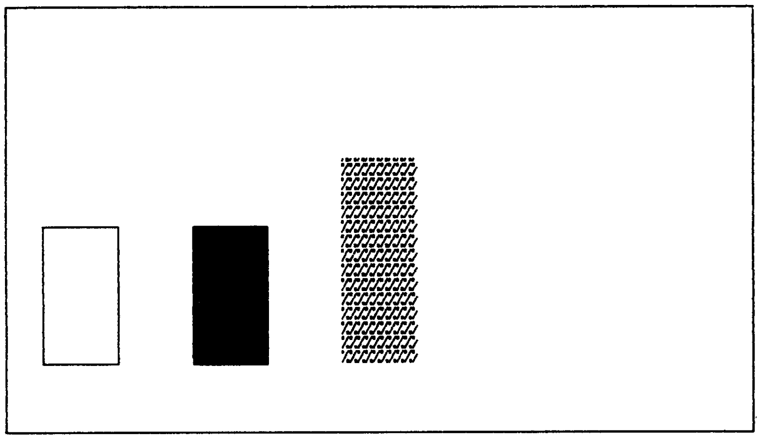
SSHADE does not use a "fill" algorithm; it shades to the shade line you define; the line itself is not a visible line on the screen.

After you call SSHADE, every point that makes up a picture object shades to the shade line. To shade a circle or arc, the shade line must pass through the center of the figure. Otherwise, the outermost point on the curve shades to the shade line and overshadows the arc as it curves inward. SSHADE shades every point in the picture object to the shade line you select even if the area to be shaded is outside the boundaries of the picture object. The following programs illustrate the effect of the shade line's location. The easiest way to understand how SSHADE works is to run the SSHADE program, SSHADE1.DEM.

The following program, named SHADE1 on your distribution volume, illustrates the effects of where the shade line is placed.

```
C   SHADE1
C   This program draws three boxes: one without shading, one with
C   shading enabled to a proper shade line, and one with the shade
C   line above the top of the box.
C
          CALL INITGR (5)
          CALL CLRTXT
          CALL CLRSCR
          CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
C
C   This BOX call creates a frame for the illustration.
C
          CALL BOX (0.,0.,1000.,625.)
C
C   Draw the first box (shading off is the initial default).
C
          CALL BOX (50.0, 100.0, 150.0, 300.0)
C
C   Draw the second box.  We select solid shading, the initial default,
C   by not specifying a pattern.  Valid ylines could be:  top or
C   bottom of box, or any horizontal line between.
C   We select the top.
C
          CALL SSHADE (300.0, 1)
          CALL BOX (250.0, 100.0, 350.0, 300.0)
C
C   Draw the third box, using "%" as the shade character and
C   using a shade line above the box.  Notice that the box looks
C   as if we had drawn it to the shade line's coordinate
C   of 400.0.
C
          CALL SSHADE (400.0, "045)
          CALL BOX (450.0, 100.0, 550.0, 300.0)
          END
```

Figure 2–9: Picture Drawn by the SHADE1 Program

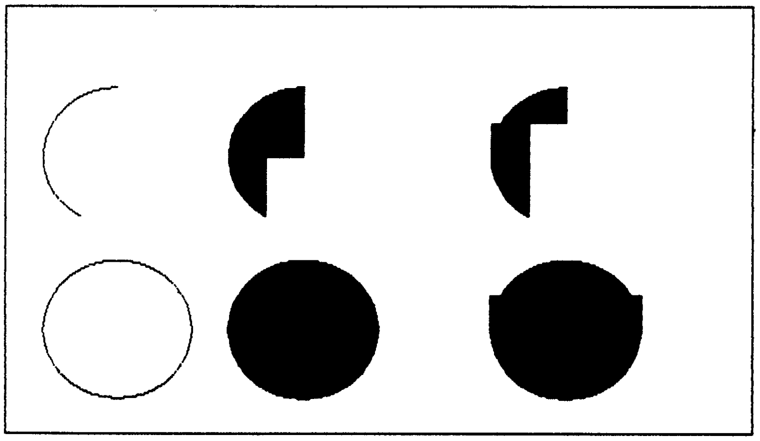


The following program, named SHADE2 on your distribution medium, illustrates the effect of where the shade line is placed, and how to shade circles properly.

```
C   SHADE2
C   This program draws three circles and three arcs.  The first
C   circle and arc are drawn with shading turned off.  The
C   second pair with the shade line set to the middle of
C   the objects, and the third pair with the shade line
C   set above the middle.
C
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
          CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
          CALL BOX (0.,0.,1000.,625.)
C
C   All angles will be interpreted in degrees.
C
          CALL SDGREE
C
C   Draw the first circle and arc, with shading off
C   as the initial default.
C
          CALL MOVE (150.0, 150.0)
          CALL CIRCLE (100.0)
          CALL MOVE (150.0, 400.0)
          CALL ARC (150.0, 150.0, 500.0)
C
C   Turn shading on and select solid fill.  Draw the circle.
C   Notice the shade line must be at the center of the
C   circle.
C
          CALL SSHADE (150.0, 1)
          CALL MOVE (400.0, 150.0)
          CALL CIRCLE (100.0)
C
C   Move the shade line up to the center of the arc.  Since the arc
C   is only 150 degrees long, the shading stops where the arc stops,
C   making the shade line obvious.
C
```

```
      CALL SSHADE (400.0, 1)
      CALL MOVE (400.0, 400.0)
      CALL ARC (150.0, 400.0, 500.0)
C
C  Now draw the third pair with a shade line above
C  the center.  Notice how the outermost points of
C  the circle and arc overshadow the inner points.
C
      CALL SSHADE (200.0, 1)
      CALL MOVE (750.0, 150.0)
      CALL CIRCLE (100.0)
      CALL SSHADE (450.0, 1)
      CALL MOVE (750.0, 400.0)
      CALL ARC (150.0, 750.0, 500.0)
      END
```

Figure 2–10: Objects Drawn by the SHADE2 Program



2.6 Marking Locations

You can mark a location on the screen by calling either the MARKER subroutine or the RELMKR (*Rel*ative *M*ar*k*e*r*) subroutine. If you call MARKER, you specify the absolute location where you want the marker to appear. If you call RELMKR, you specify the x– and y–distances from the current location of the graphic cursor. RGL/11 uses the location you specify as the center of the marker.

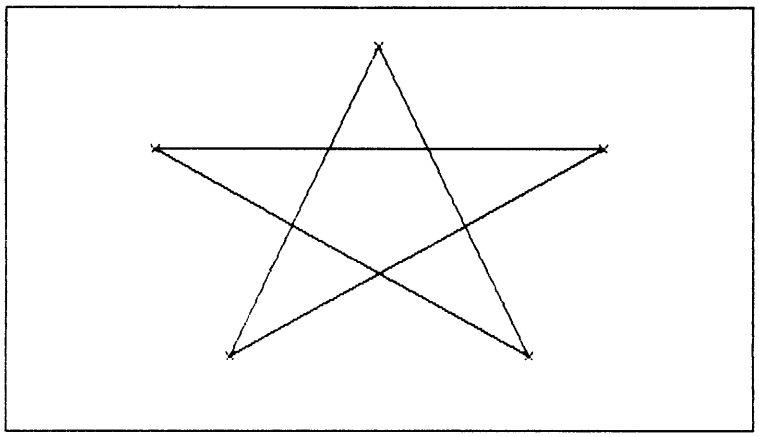
A marker can be a simple dot or any of the symbols listed in the MARKER subroutine, Chapter 5.

The following program, named MARK on your distribution volume, places a marker at each vertex of a star. (The program is the STAR program listed in Section 2.3 plus additional statements to mark the vertices.) Figure 2–11 shows the picture created by MARK.

```
C  MARK
C  This program draws a star by calling POLYLN
C  and then marks the vertices by calling MARKER.
```

```
C
        DIMENSION XSTAR (5), YSTAR (5)
        DATA XSTAR /500.0, 700.0, 200.0, 800.0, 300.0/
        DATA YSTAR /560.0, 110.0, 410.0, 410.0, 110.0/
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
        CALL BOX (0.,0.,1000.,625.)
C
C   Move the graphic cursor to the starting location.
C
        CALL MOVE (300.0, 110.0)
C
C   Draw the star.
C
        CALL POLYLN (5, XSTAR, YSTAR)
C
C   Now mark the vertices with an "X" (number 5).
C
        DO 10, I = 1,5
        CALL MARKER (5, XSTAR(I), YSTAR(I))
10      CONTINUE
        END
```

Figure 2–11: Picture Drawn by the MARK Program



2.7 Retrieving Location Coordinates

When you want to know the coordinates of a location, you can use either the LOCATE or GETLOC (*Get Loc*ation) subroutines. GETLOC returns the coordinates of the current location of the graphic cursor, storing them in variables named X and Y.

LOCATE creates a special locator cursor that you can interactively move around on the screen to the location or locations you want. The cursor is a large white crosshair with a blinking diamond–shaped polygon in the center. You move the cursor by pressing the arrow keys on the top right of the terminal keyboard. When you first call LOCATE, the cursor moves a distance of ten pixels every time you press an arrow key. To slow it down for more precise placement, press the PF3 key; thereafter the cursor moves one pixel at a time. To speed it up again, press the PF4 key. When you have located the cursor at a point whose coordinates you want, press any key but an arrow key, a SHIFT, ESCAPE, or DELETE key. LOCATE puts the point's coordinates in the X and Y variables. By placing a LOCATE call along with a READ or TYPE call inside a FORTRAN loop, you can read and store the coordinates of several locations.

In the following program, named LOCAT on your distribution volume, you can move the locator cursor repeatedly and see the coordinates you want printed out on the screen. To end the program, you type the "X" key.

```
C   LOCAT
C   This program uses the locator cursor and the LOCATE
C   subroutine to retrieve the coordinates of a point
C   or points, as the user chooses.
C
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
          CALL SWINDO (0., 0., 1000., 625.)
          CALL BOX (0.,0.,1000.,625.)
          CALL LINETX (1,1,'Type uppercase ""X"" to exit.')
C
C   Define X and Y to 500. and 300. respectively and call LOCATE.
C
          X=500.
          Y=300.
10        CALL LOCATE (X,Y,KEY)
C
C   The locator cursor is at location (500.,300.).  You
C   can begin to move it using the arrow keys.  Select
C   the locations whose coordinates you want by typing
C   any key but "X", the DELETE key, or the arrow keys.
C   The coordinates will be printed on the screen.  To
C   end the loop and close the program, type an upper-
C   case "X."
C
          TYPE 100, X, Y, KEY
          IF (KEY.NE."130) GO TO 10
   100    FORMAT (' ', 'X=        ', F6.0, 5X, 'Y=        ', F6.0,5X,
          1'KEY=    "',O3)
          END
```

Other LOCATE examples are in the OBJECT program in Section 2.1 and in Chapter 5, the LOCATE subroutine.

2.8 Labeling the Picture

You can use graphic text to label picture objects using either the LINETX or TEXT subroutines. The LINETX (*Line Text*) subroutine displays text strings on the screen, beginning at the row and column location you specify. LINETX can display text strings of alternate character sets, but it always uses characters in standard size only. To get a more varied display of graphic text, use the TEXT subroutine. TEXT will use the character size, character set, and location you have specified in preceding calls to MOVE, STXSIZ (*S*et *T*e*x*t *Size*), and the LCHRST (*L*oad *Char*acter *Set*) and SCHRST (*S*et *Char*acter *Set*) subroutines.

Using the MOVE subroutine, you can place the graphic cursor at the location where you want the text to start. The current graphic cursor location marks the top left corner of the first character cell of the TEXT string. TEXT leaves the graphic cursor at the end of the string.

Using the STXSIZ subroutine and then TEXT, you can display graphic characters in sixteen different heights and sixteen widths.

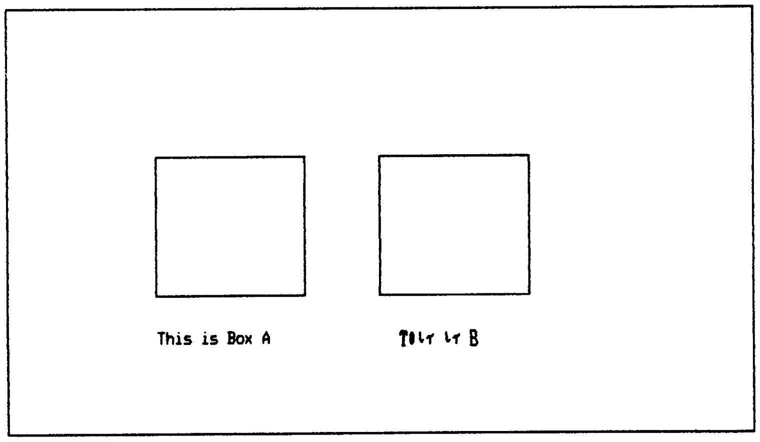
In addition to the standard character set, the Greek character set is also available on your distribution volume. It is in a file called GREEK.FNT. To use the Greek character set, you first load it into character set 1, using the LCHRST (*L*oad *Ch*aracter *S*e*t*) subroutine. Then you enable that set by the SCHRST (*S*et *Ch*a*r*acter *S*e*t*) subroutine.

When you want to return to the standard character set, you need only call SCHRST with a character set argument of 0, for the standard set. You do not need to load the standard set, because it is always in memory.

The following program, named LABEL1 on your distribution volume, illustrates labeling subroutines.

```
C     LABEL1
C     This program draws two boxes and uses the
C     English character set to label one and the
C     Greek character set to label the other.
C
            CALL INITGR (5)
            CALL CLRSCR
            CALL CLRTXT
            CALL SWINDO (0.,0.,1000.,625.)
            CALL BOX (0.,0.,1000.,625.)
C
            CALL BOX (200.,200.,400.,400.)
            CALL MOVE (200.,150.)
            CALL TEXT ('This is Box A')
C
            CALL BOX (500.,200.,700.,400.)
            CALL MOVE (525.,150.)
            CALL LCHRST (1,'GREEK.FNT',)
            CALL SCHRST (1)
            CALL TEXT ('This is B')
            END
```

Figure 2–12: Picture Drawn by the LABEL1 Program



2.9 Using Writing Modes

When drawing graphic objects on the screen, you can draw in any of the following writing modes:

Mode	Effect
Overlay	covers objects transparently; the underlying pattern is still visible.
Replace	covers objects opaquely; the underlying pattern is no longer visible.
Erase	erases selected graphic objects from the screen, and replaces them with the screen color.
Complement	highlights intersecting graphic objects or erases objects by replacing them with the color of the underlying object.
Reverse	interchanges screen color and drawing color.
No–Reverse	negates the reverse writing mode.
Initialize	re–establishes overlay mode and no–reverse mode as the current writing mode; that is, it re–establishes the initial conditions in one step, not two.

You enable a writing mode by calling the SWMODE (Set Writing *Mode*) subroutine. That writing mode stays in effect until you call SWMODE again or INITGR. When you first call INITGR, at the beginning of a graphic program, the overlay mode is in effect, with reverse writing mode turned off.

To understand the different modes, you need to understand a little about how the terminal draws picture objects and graphic text on the screen. The screen is composed of "*pixels*," which are "picture elements", the smallest displayable elements on the screen. When the terminal is turned on, they are all displayed in GRAY0, the default screen color.

When you set *overlay* mode on and draw a non–solid figure on top of a line, the line is still visible. When you set *replace* mode on and draw a figure (of either a solid or non–solid pattern) on top of a line, the line will no longer be visible; all pixels in that line will be redrawn in the figure's drawing color.

If *erase* mode is on and you redraw the line, you "erase" it because you replace the pixels' color with the screen color. Suppose, however, that a circle overlaid the line and you wanted to erase the circle but not the line (as the erase mode would do); the *complement* mode accomplishes that. To use complement mode to erase, you must invoke complement mode, draw the figure, and then draw it again. The second time the figure is drawn, it will be erased (but any underlying object remains).

Most picture objects require just one drawing color. Some, however, require both a drawing color and the screen color. For example, in a line of dashes, the terminal draws the dash in the drawing color and draws the space between the dashes in the screen color.

Writing graphic text also requires both screen color and drawing color. Graphic text writes a character into a *"character cell"*, a matrix of 10 pixels high, eight pixels wide. The graphic characters are written in a row/column format. For example, the character cell for the letter T looks like this:

```
0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

where 1 is the drawing color and 0 is the screen color

In sum, the writing modes affect how each pixel changes as you draw graphic objects on the screen, particularly when the graphic object being drawn intersects with another graphic object already on the screen. The writing modes govern whether or not a pixel will be overwritten by the new color, will remain unchanged, or will revert to a default color. It is by this process that you can, in effect, place an object "in front of" or "behind" another object, and can achieve other visual effects.

The next sections describe the writing modes in detail, so that you will know which writing mode is best suited for your purposes.

2.9.1 Overlay Mode

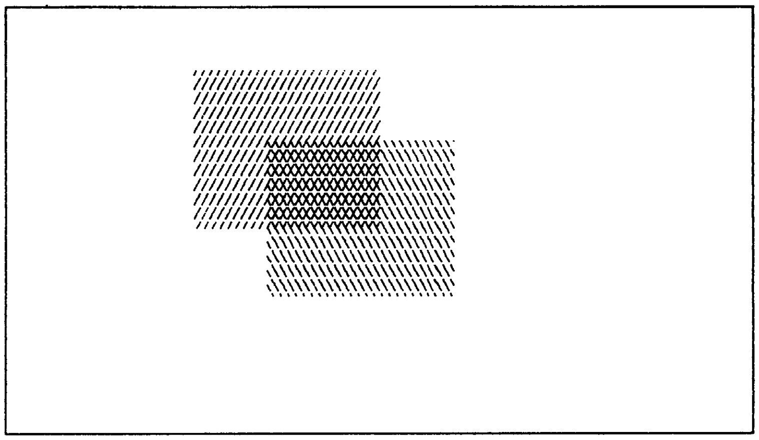
The effect of overlay mode is to draw a graphic object while having an underlying pattern remain visible also. This mode gives you the most complete picture of two intersecting graphic objects. The effect of overlay mode is only noticeable for non–solid shade patterns.

The following FORTRAN program, named OVERLY on your distribution volume, shows an example of writing with overlay mode. Figure 2–13 shows the picture created by the example.

```
C  OVERLY
C  This program draws two boxes in overlay mode.  Each box is
C  shaded with a different character to show you how they overlap.
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
        CALL BOX (0.,0.,1000.,625.)
        CALL SWMODE ('OV')
C
C  Set shading to the '/' character and draw a box.
C
        CALL SSHADE (300.0, "57)
        CALL BOX (250.0, 300.0, 500.0, 525.0)
```

```
C
C   Set shading to the '\' character and draw a box.
C
          CALL SSHADE (200.0, "134)
          CALL BOX (350.0, 200.0, 600.0, 425.0)
          END
```

Figure 2–13: Example of Drawing with Overlay Mode



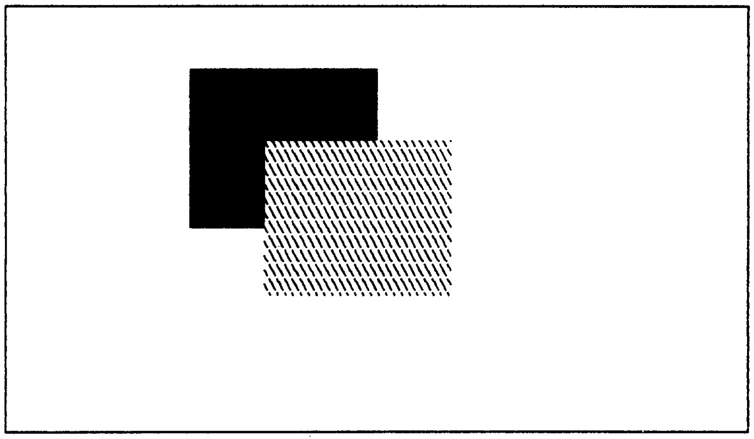
2.9.2 Replace Mode

When you draw in replace mode, you completely cover any underlying pattern with the new drawing color and drawing pattern. The effect is that the more recently drawn image appears to be "in front of" the other, and appears opaque; its pattern is unaffected by the underlying image.

The following FORTRAN program, named REPLAC on your distribution volume, shows how to write in replace mode. Figure 2–14 shows the picture created by this program.

```
C   REPLAC
C   This program draws two boxes that overlap.  The boxes are
C   written in the replace writing mode.
C
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
          CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
          CALL BOX (0.,0.,1000.,625.)
          CALL SWMODE ('RE')
C
C   Draw the first box with a solid fill (default).
C
          CALL SSHADE (300.0, )
          CALL BOX (250.0, 300.0, 500.0, 525.0)
C
C   Draw the second box with the shading character '\'.
C
          CALL SSHADE (200.0, "134)
          CALL BOX (350.0, 200.0, 600.0, 425.0)
          END
```

Figure 2–14: Example of Drawing with Replace Mode



An example of replace mode in combination with reverse mode is given in Section 2.9.5, Reverse Mode.

2.9.3 Erase Mode

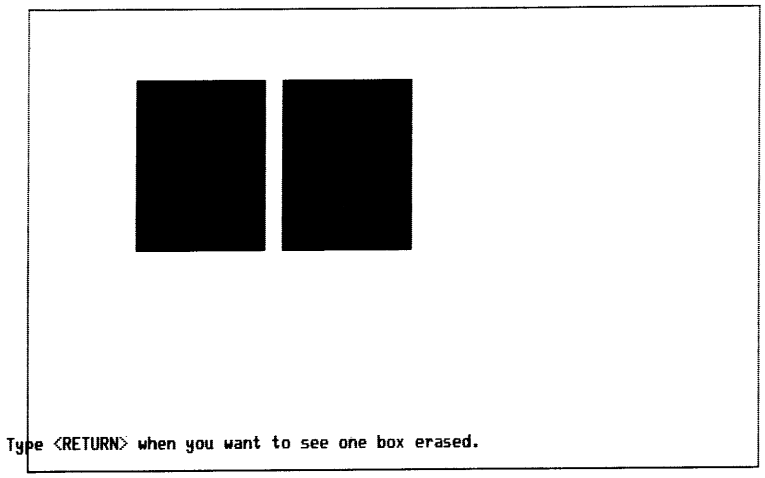
Erase mode causes the pixels of a graphic object to revert to the screen color, no matter what color they had. Thus, when you want to selectively erase graphic objects from the screen, you redraw them with erase mode enabled. The writing mode you used to draw the object originally in no way affects the results of erase mode. You can have used overlay, replace, or complement mode; the pixels are returned to the screen color.

The following FORTRAN program, named ERASE on your distribution volume, shows an example of writing with erase mode. Figure 2–15 shows the picture created by the program.

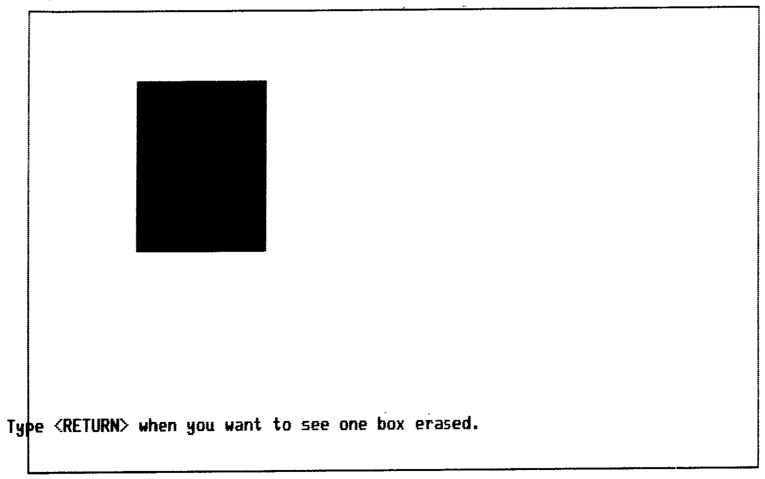
```
C   ERASE
C   This program draws two shaded boxes.  When you type the
C   <RETURN> on the keyboard, the second box is erased.
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
        CALL BOX (0.,0.,1000.,625.)
C
C   Set solid shading and draw two boxes.
C
        CALL SSHADE (300.0, )
        CALL BOX (150.0, 300.0, 325.0, 525.0)
        CALL BOX (350.0, 300.0, 525.0, 525.0)
C
C   Wait until the user types <RETURN>.
C
        TYPE 100
100     FORMAT (' Type <RETURN> when you want to see one box erased.')
        ACCEPT 200
200     FORMAT (' ')
C
```

```
C  Now erase the second box.  Remember that shading is
C  still enabled, so the shaded box is erased.
C
       CALL SWMODE ('ER')
       CALL BOX (350.0, 300.0, 525.0, 525.0)
       END
```

Figure 2–15: Example of Drawing with Erase Mode



```
Type <RETURN> when you want to see one box erased.
```



```
Type <RETURN> when you want to see one box erased.
```

2.9.4 Complement Mode

Using complement mode, you can erase an intersecting object and get back the original, underlying, object (not the screen color as in erase mode). You can also use complement mode to highlight graphic objects that intersect other graphic objects.

When you use complement mode to highlight graphic objects, it "complements" the color number of each affected pixel according to the following chart:

Pixel Color Number Before the Complement Mode Call	Pixel Color Number After the Complement Mode Call
0	3
1	2
2	1
3	0

The drawing–color number has no effect on the resultant color in complement mode. Only the current color number of each affected pixel is significant. To understand the distinction between color number and color name, read Section 2.10, Selecting Gray Shades or Color.

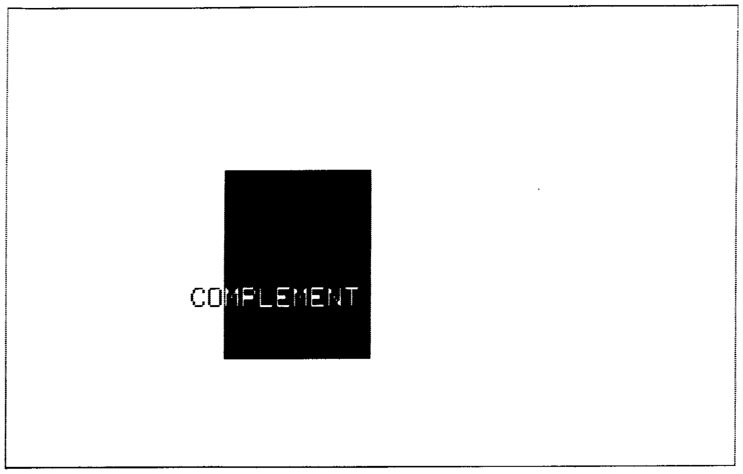
The color actually displayed on the screen depends upon what color has been associated with the pixel's color number. For example, if GRAY2 is associated with color number 2, and GRAY1 associated with 1, then a pixel with color number 2 which is rewritten in complement mode has its color number complemented to 1 and so its new color is GRAY1. However, if GRAY3 is associated with color number 1, the pixel with color number 2 would receive GRAY3 color, because the color *number* is complemented, not the color itself.

Notice again that, with complement mode, the drawing color of the object being drawn is not used at all when you draw; rather the pixels are colored by the complement of the color *number* they already have.

The following program, named COMPLE on your distribution volume, shows how complement mode highlights graphic objects that intersect. Figure 2–16 shows the picture created by this program.

```
C   COMPLE
C   This program draws a shaded box and then writes
C   graphic text over it in complement mode.
C
         CALL INITGR (5)
         CALL CLRSCR
         CALL CLRTXT
         CALL SWINDO (0.,0.,1000.,625.0)
         CALL BOX (0.,0.,1000.,625.)
C
C   Set solid shading for the box to be drawn.
C
         CALL SSHADE (150., )
C
C   Draw the box, using overlay mode and GRAY3
C   (the initial defaults).
C
         CALL BOX (300.,150.,500.,400.)
C
C   Now write text across the box in complement mode.
C
         CALL SWMODE ('CO')
         CALL MOVE (250.,250.)
         CALL STXSIZ (2,)
         CALL TEXT ('COMPLEMENT')
         END
```

Figure 2–16: Example of Drawing with Complement Mode



Complement mode can also erase graphic objects. In erase mode, the pixels that comprise the object revert to the screen color. Thus if an object intersects a second object that is being drawn in erase mode, the first object will have "holes" in it when the erase mode operation is completed (because erase mode returns the screen color). However, if you use complement mode, the first object will remain intact after the second object is erased. Use complement mode to erase overlapping objects so as not to erase underlying objects.

To erase a graphic object when writing in complement mode, you must:

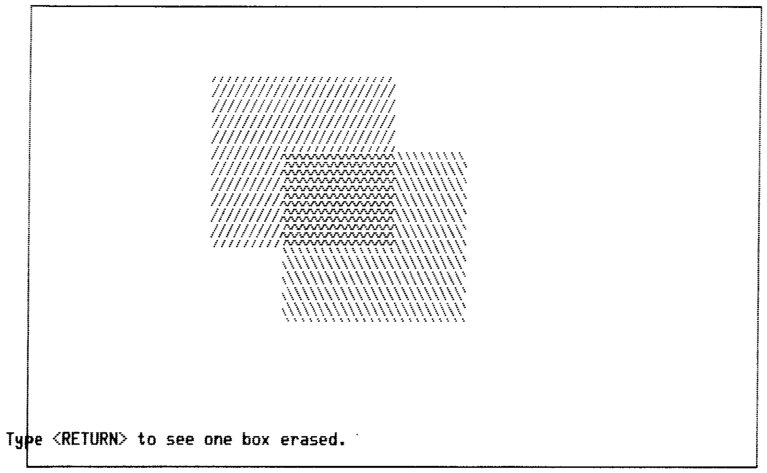
1. Draw the object in complement mode when you first draw it.
2. Redraw the object with complement mode when you want to erase it. You must redraw it exactly as it was first drawn; for example, redraw a line in the same direction as the original line, as well as drawing it between the same two points.

The following program, named ERASEC on your distribution volume, shows an example of erasing with complement mode.

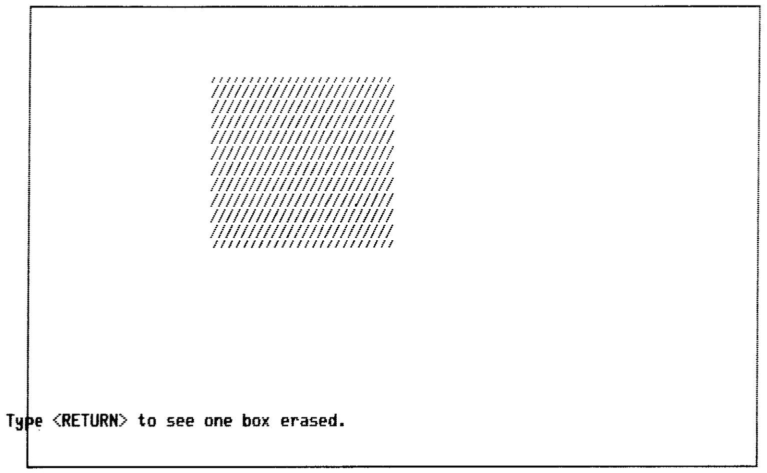
```
C   ERASEC
C   This program draws two shaded boxes; the second box is
C   drawn using complement mode.  When you type <RETURN> on
C   the keyboard, the second box is erased using complement
C   mode.
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL SWINDO (0.0,0.0,1000.0,625.0)
        CALL BOX (0.,0.,1000.,625.)
C
C   Shade with the '/' character and draw the first box.
C
        CALL SSHADE (300.0,"57)
        CALL BOX (250.0,300.0,500.0,525.0)
```

```
C
C   Shade the second box with the '\' character and draw
C   it in complement mode.
C
         CALL SWMODE ('CO')
         CALL SSHADE (200.0,"134)
         CALL BOX (350.0,200.0,600.0,425.0)
C
C   Wait until the user types <RETURN>.
C
         TYPE 100
100      FORMAT (' Type <RETURN> to see one box erased.')
         ACCEPT 200
200      FORMAT (' ')
C
C   Now erase the second box by redrawing the box.
C   Complement mode is still enabled.
C
         CALL BOX (350.0,200.0,600.0,425.0)
         END
```

Figure 2–17: Example of Erasing with Complement Mode



```
Type <RETURN> to see one box erased.
```



```
Type <RETURN> to see one box erased.
```

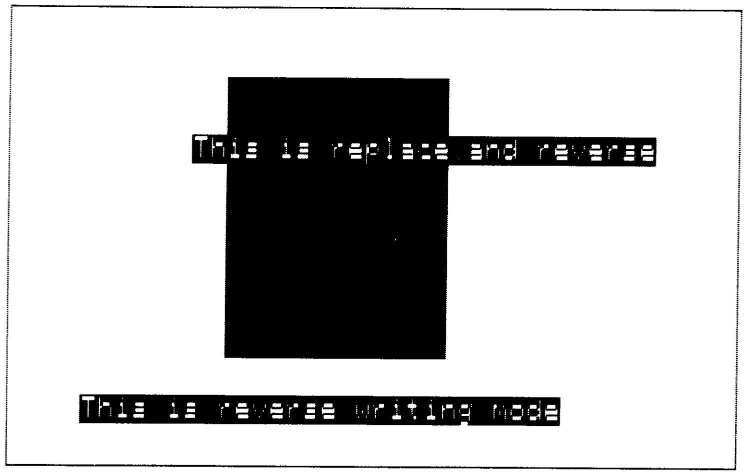
2.9.5 Reverse Mode

Reverse writing mode displays graphic objects in reverse image: the screen color number becomes the drawing color number and vice versa. With a solid shade pattern, nothing is drawn in reverse mode; therefore, use reverse mode with patterns other than solid patterns. You can combine reverse mode with overlay, replace, complement or erase writing modes.

The following program, named REVERS on your distribution volume, shows how to write using reverse mode. Figure 2–18 shows the picture created by the program.

```
C   REVERS
C   This program writes a message in reverse writing mode.  The
C   program then shows an example of combining reverse and replace
C   writing modes.
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
        CALL BOX (0.,0.,1000.,625.)
        CALL STXSIZ (2, )
C
C   Set the reverse writing mode and write a graphic text message.
C
        CALL SWMODE ('RV')
        CALL MOVE (100.0, 100.0)
        CALL TEXT ('This is reverse writing mode')
C
C   Now combine reverse mode with replace mode and see what happens.
C   First, draw a box with solid shading and no reverse.
C
        CALL SWMODE ('NR')
        CALL SSHADE (150.0, )
        CALL BOX (300.0, 150.0, 600.0, 525.0)
C
C   Now write text across the box in reverse and replace modes.
C
        CALL SWMODE ('RV')
        CALL SWMODE ('RE')
        CALL MOVE (250.0, 450.0)
        CALL TEXT ('This is replace and reverse')
        END
```

Figure 2–18: Example of Drawing with Reverse Mode



2.9.6 No–Reverse Mode

The no–reverse mode cancels the effect of the reverse mode. In no–reverse writing mode, the drawing color is the drawing color, the screen color is the screen color; they are not reversed. The no–reverse writing mode and the overlay mode are in effect after an INITGR call.

2.9.7 Initialize Mode

The initialize mode re–establishes the no–reverse and the overlay writing modes. One call to this writing mode controls two writing mode features. Thus this mode is a "labor–saving" call.

2.10 Selecting Gray Shades or Color

The VT125 can display four shades of gray on its black–and–white monitor. If you attach a color monitor, it can simultaneously display four colors on the color monitor.

The four shades of gray on the VT125 black–and–white monitor are fixed; you cannot change their intensity. They are:

Color Names	Description
GRAY0	Dark, the absence of light on the screen (this is the usual screen color)
GRAY1	Dark gray
GRAY2	Light gray
GRAY3	White

You use the SCOLOR (*Set Color*) subroutine to control the colors RGL/11 uses in the picture–drawing subroutines and in one data–plotting subroutine, PPOINT. (For other data–plotting subroutines, see Section 3.7, Selecting Gray Shades or Color.) SCOLOR's format is:

SCOLOR ({'name'}, number)

The *color names* in the above list are the values you supply for the 'name' argument, and *color numbers* 0 through 3 are the values for the number argument. The numbers 1, 2, and 3 control the current drawing color; number 0 controls the screen color.

SCOLOR has three functions:

1. It associates the specified color name with the color number.
2. If you specify a color number of 1, 2, or 3, SCOLOR changes the current *drawing color* number to the specified color number, and also changes any picture image drawn with that color number to the new color associated with it. For example, if you associate GRAY2 with color number 3, then all subsequent objects are drawn in GRAY2, and any previously drawn objects that were drawn with color number 3 are now drawn in GRAY2.
3. (this function of SCOLOR is more rarely used) If you specify 0 as the color number, SCOLOR changes the *screen color* to the color now associated with 0. But it also establishes that color as the drawing color, so you need to call SCOLOR immediately to establish a new drawing color.

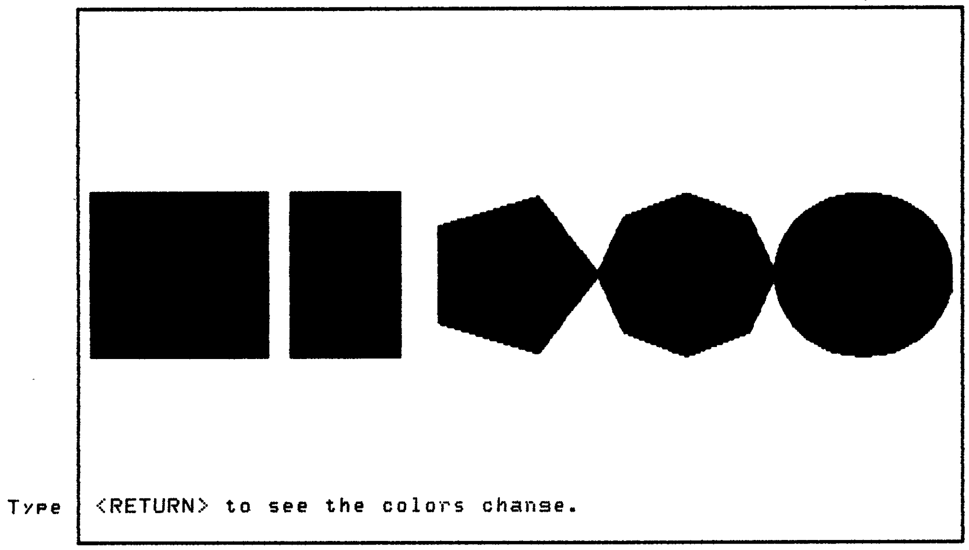
One color name can be associated with several color numbers, but each number can reference only one color name at a time. By an INITGR call, GRAY0 is associated with 0 (the initial screen color), GRAY1 with 1, GRAY2 with 2, and GRAY3 with 3 (the drawing color).

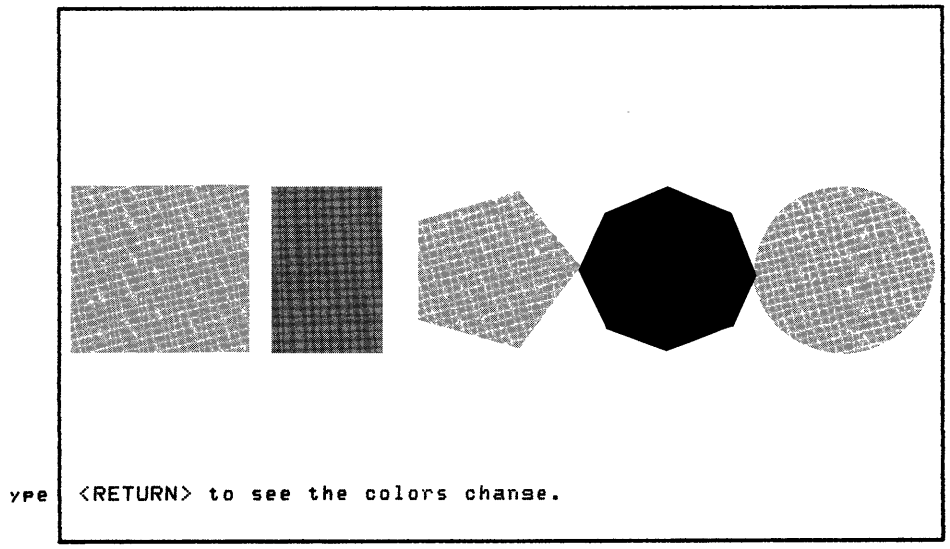
The following program, HILITE on your distribution volume, shows how the shade changes when you associate a new shade with a color number.

```
C   HILITE
C   This program shows how the VT125 can highlight objects
C   during a program.  The program draws five objects in
C   GRAY3, but with different color numbers.  When the
C   user types the RETURN key, the program reassigns
C   colors to color numbers so that three objects immediately
C   change to GRAY1 and one to GRAY2.
C
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
          CALL SWINDO (0.,0.,1000.,625.)
          CALL BOX (0.,0.,1000.,625.)
C
C   Enable shading, using the default line patterns.
C
          CALL SSHADE (300.,)
C
C   Draw three objects in color number 1, using GRAY3.
C
          CALL SCOLOR ('GRAY3',1)
          CALL BOX (25.,200.,225.,400.)
          CALL MOVE (500.,300.)
          CALL POLYGN (5,100.)
          CALL MOVE (900.,300.)
          CALL CIRCLE (100.)
```

```
C
C  Draw the fourth object in color number 2, using GRAY3.
C
         CALL SCOLOR ('GRAY3',2)
         CALL BOX (250.,200.,375.,400.)
C
C  Draw the fifth object in color number 3, using GRAY3.
C
         CALL SCOLOR ('GRAY3',3)
         CALL MOVE (700.,300.)
         CALL POLYGN (8,100.)
C
C  When the user types <RETURN>, the program reassigns color
C  number 1 to GRAY1, and 2 to GRAY2, causing the first four
C  objects to be changed.
C
         TYPE 100
100      FORMAT (' Type <RETURN> to see the colors change.')
         ACCEPT 200
200      FORMAT (' ')
         CALL SCOLOR ('GRAY1',1)
         CALL SCOLOR ('GRAY2',2)
         END
```

Figure 2–19: Example from the HILITE Program





An optional color monitor can be attached to the back of a VT125 black–and–white monitor. For more information on attaching a color monitor to a VT125, see the *VT125 Graphics Terminal User Guide.* When you add a color monitor, the VT125 displays four colors on the color monitor screen while displaying the same images in black and white on its own screen.

The gray shades appear on the color monitor as follows:

GRAY0 appears as dark
GRAY1 appears as blue
GRAY2 appears as red
GRAY3 appears as green

Chapter 3
Data Plotting

Seven RGL/11 subroutines are data–plotting subroutines. They enable you to plot data on two–dimensional graphs, using data that you have collected before the plot is made or data you plot point by point. Thus data can be entered from stored files or on a point–by–point basis.

The data plotting subroutines are:

DPAPER	— (*D*raw *Paper*) creates a grid on which to plot the data.
LNAXIS	— (*L*abel *N*umeric *Axis*) labels and scales an x– or y–axis, labeling the cell boundaries on the axis with numeric values, and labeling the axis.
LTAXIS	— (*L*abel *T*ext *Axis*) labels and scales an x– or y–axis, labeling the cell boundaries on the axis with alphanumeric values, and labeling the axis.
LFIXED	— (*L*ocate *Fixed*) displays an interactive cursor on top of the data line of a graph, and returns pointers into the two arrays passed to LFIXED for up to ten points on the data path of a graph in the units of the graph's axes.
LFREE	— (*L*ocate *Free*) displays an interactive cursor anywhere within the graph's boundaries, returns the x– and y–coordinates of a point anywhere in the graph in the units of the graph's axes.
PDATA	— (*P*lot *Data*) plots arrays of data on the grid you select, using data that has already been collected.
PPOINT	— (*P*lot *Point*) adds single data points to an existing graph.

The data–plotting "environment" is different from the picture–drawing environment in that the defaults for some terminal settings are different. The data–plotting environment is created by either the PDATA or the DPAPER subroutine.

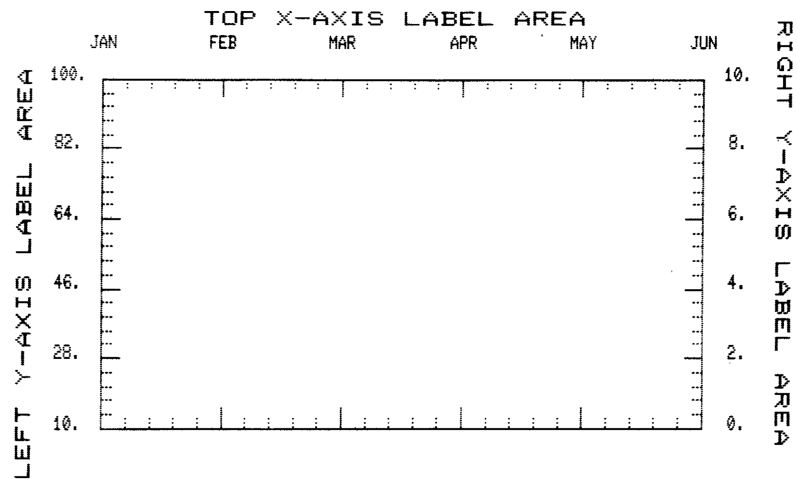
NOTE

A certain set of parameters are changed in data plotting operations by DPAPER or PDATA and are not reset. The parameters are:

- color assignment (color name to color number)
- current drawing color
- line pattern
- world coordinates
- shading
- text size
- writing mode

To reset these parameters, use the appropriate subroutine: SCOLOR, SLNPAT, SWINDO, SSHADE, STXSIZ, or SWMODE.

Figure 3–1: Sample Layout of a RGL/11 Graph



The simplest way to create a graph uses just one RGL/11 subroutine, PDATA. That subroutine is described in Section 3.1, Data Plotting with One Subroutine.

Using other subroutines, you can plot on logarithmic grids and further customize the data–plotting process.

The sections in this chapter are:

3.7 Selecting Gray Shades or Color
3.8 Plotting the Data

NOTE

As in Chapter 2, the sample programs in this chapter are also files on your distribution volume and use the file extension of .DEM. This extension makes it easy to get a directory listing of all demo programs. When you use the RGLLNK indirect command file to run them, you need to supply both the file name and .DEM extension. (For RGLLNK on RT–11 systems, see Section 4.3.2.2; on RSX–11M systems, see Section 4.4.2.2.)

The file names, but not extensions, are given in the following sections.

3.1 Data Plotting with One Subroutine

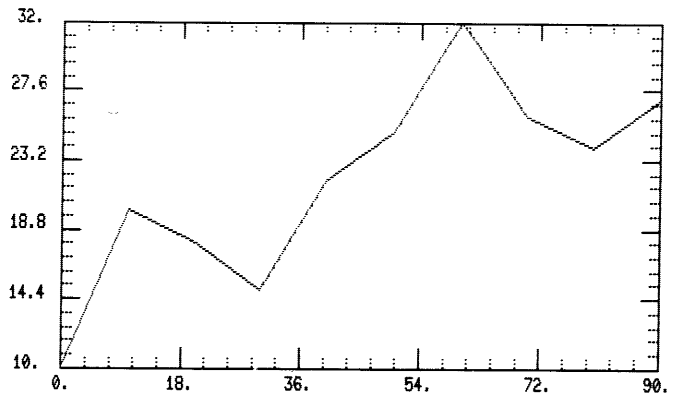
You can use just one subroutine, PDATA, to create a graph. If you supply just two of its ten arguments (the number of coordinate pairs to be plotted and the name of the array containing the coordinate values for the y–axis) and you supply commas to delineate the other arguments, PDATA will draw a graph and plot the data on it. The graph will have these characteristics:

- the graph is linear on both the x– and y–axes.
- the graph is "ungridded;" that is, its axes have long tickmarks at the cell boundaries and short tickmarks at the subcell boundaries.
- the axes have no labels, but the cell markers have numeric labels. The numeric labels are autoscaled, and have rounded numbers based on the data. The bottom x–axis and left y–axis are scaled.
- each axis has five cell divisions, and each cell has five subcell divisions.

The following program, named XYPLOT on your distribution volume, illustrates the PDATA subroutine with only three argument values supplied. Figure 3–2 shows the plot generated by the example.

```
C  XYPLOT
C
      DIMENSION XDATA (10), YDATA (10)
      DATA XDATA/0.0, 10.,20.,30.,40., 50.0,60.,70.,80.,90.0/
      DATA YDATA/10.,20.,18.,15.,22.,25.,32.,26.,24.,27./
C
      CALL INITGR (5)
      CALL CLRTXT
      CALL CLRSCR
      CALL PDATA (10, XDATA, YDATA,,,,,,,)
      END
```

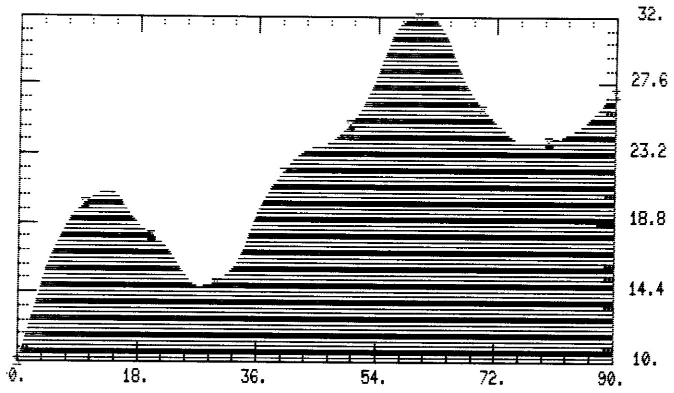
Figure 3–2: Example of Basic, One–Call Data Plotting



Using the same program and changing just some arguments in the PDATA call, you can introduce variations. For example, the following PDATA call gives the graph shown in Figure 3–3.

```
CALL PDATA (10,XDATA,YDATA,'R','GRAY1',9,5,,TRUE.,,TRUE.,)
```

Figure 3–3: Variations of the One–Call Data Plotting Subroutine



Other arguments in the PDATA subroutine change the line pattern, the marker, and so forth. (You can also use PDATA with only the DPAPER subroutine to create logarithmic graphs, or with DPAPER, LNAXIS, and LTAXIS to create labeled graphs. See Section 3.8.1, Plotting Data from Stored Files.)

3.2 Creating the Graph Paper

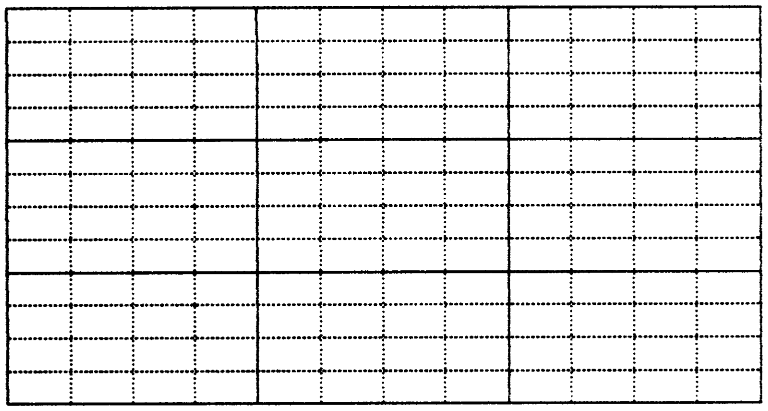
Using the DPAPER subroutine, you can draw gridded or ungridded graph paper, and you can create two types of graph paper, linear and logarithmic. You establish these characteristics by arguments in the DPAPER subroutine. When you establish the number of major divisions (cells) of the x–axis and y–axis and you intend to use rounded values for the graph, be sure to supply values for the cell arguments which are multiples of 2 or 5 to allow the rounded–number algorithm to function properly. (See Section 3.3.1, Scaling in Rounded Numbers.)

3.2.1 Gridded and Ungridded Graph Paper

Gridded graph paper consists of a series of parallel lines drawn across the entire grid. Ungridded graph paper has tickmarks on the top and bottom x–axes and on the left and right y–axes.

Each axis has major divisions called *cells*. The number of cells is established by the DPAPER subroutine. Cell boundaries are drawn in either solid lines across the graph or long tick marks, depending upon whether the graph is gridded or ungridded. Subcells are subdivisions of a cell. They are also created by the DPAPER subroutine. They are drawn either in dotted lines across the graph or in short tickmarks. See Figure 3–2 for an example of ungridded graph paper, and Figures 3–4 and 3–5 for examples of gridded paper with cell and subcell lines.

Figure 3–4: Gridded, Linear Graph Paper



3.2.2 Linear and Logarithmic Graph Paper

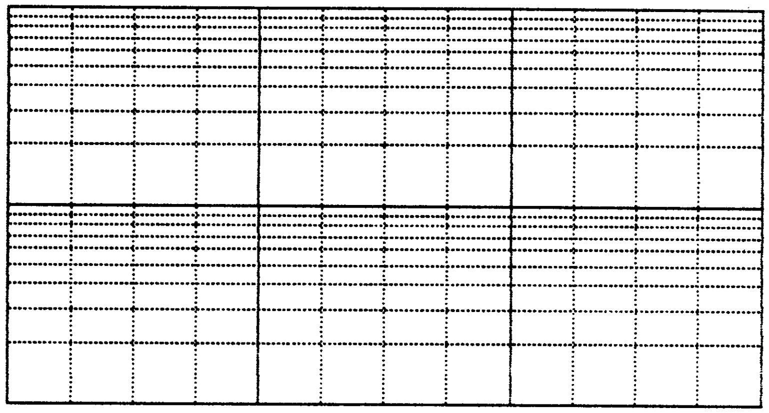
Linear graph paper consists of uniformly spaced horizontal and vertical lines. You can space the lines as close together or as far apart as you want, but horizontal and vertical lines are always equidistant from each other.

Logarithmic graph paper, on the other hand, consists of horizontal and vertical lines that are not equally spaced. *Logarithms* are numbers that indicate the power to which a fixed base must be raised to produce a given number. DPAPER creates logarithmic graph paper that uses a logarithmic scale with a fixed base of 10, that is, it uses *common logarithms*, not logarithms to the base e.

The lines along a logarithmic axis are not separated by equal distances because logarithms do not increase linearly in value. The distances along the logarithmic axis reflect the logarithmic ratio and the distances along the linear axis represent the linear ratio.

The logarithmic scale can be on the horizontal or the vertical axis or on both. When only one axis is used to represent a logarithmic scale, the graph paper is called *semi–log graph paper.* Semi–log paper is usually linear along the horizontal axis and logarithmic along the vertical axis. Figure 3–5 is configured in this way.

Figure 3–5: Semi–Log Graph Paper

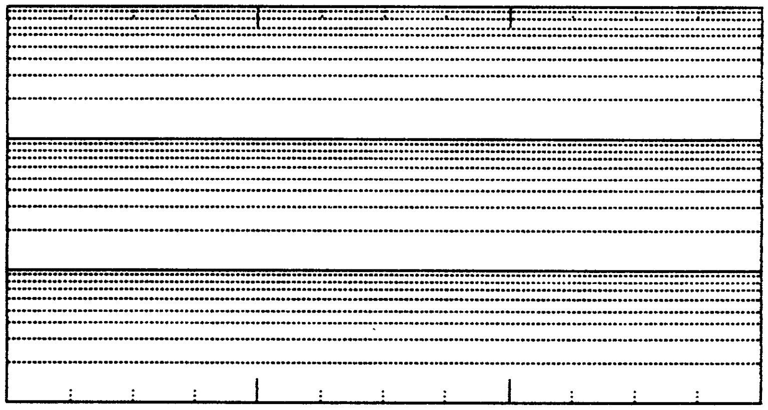


Semi–log graph paper can be either single–cycle or multi–cycle (as in Figure 3–5). Single–cycle paper shows only one logarithmic scale, the numbers in a single decade (a single power of 10). Thus, single–cycle paper can be used to plot the logarithms of numbers from 1 to 10, 10 to 100, 100 to 1000, and so on.

The spacing of numbers in a logarithmic scale is constant only for that scale. Therefore, to plot logarithms of a wider range than a single power of 10, you must add additional cycles to the paper. Figure 3–6 is an example of 3–cycle semi–log graph paper. On such paper, if the first cycle represents the logarithmic scale of 1 through 10, then the second cycle represents the logarithms of 10 through 100, and the third cycle represents the logarithms of 100 through 1000. On logarithmic axes, one cell equals one decade.

The arguments of the DPAPER subroutine enable you to choose the scale (linear or logarithmic) and the density of lines in the graph. You can select one scale for the x–axis, and another for the y–axis; you can create two y–axis scales if they are both based on the same x–axis scale. When you create multi–cycle graph paper, you can plot data with respect to either the left or right y–axis. This is described in Section 3.3, Scaling and Labeling the Axes.

Figure 3–6: 3-Cycle Semi-Log Paper



3.3 Scaling and Labeling the Axes

The LNAXIS subroutine enables you to create one or more linear or logarithmic scales on the grid DPAPER created.

3.3.1 Scaling in Rounded Numbers

"Scaling" an axis defines the range, the first and last values for that axis. When you plot numeric data, you can scale the axis either with the exact values of the given data or with a set of rounded numbers that RGL/11 can derive from the given data.

Rounded numbers are numbers that allow you to interpolate values by sight. For example, it is easier to estimate the midpoint between 0 and 10 than it is between 2.0115 and 9.3714, making 0 and 10 easier to work with. RGL/11 calculates rounded numbers according to the scale on which you want to plot your data.

For linear scales, it calculates rounded numbers by finding multiples of 2, 5, or 10 (multiplied by the appropriate power of 10) that are less then or equal to the lowest value specified and greater than or equal to the highest value specified. For example, if the data's minimum and maximum values were 11 and 22, the resulting range of rounded numbers is 10 through 22. These figures result because 10 is the nearest multiple of 2, 5, or 10 that is less than 11, and 22 is the nearest multiple of 2 that is greater than or equal to 22. In another example, if the data's minimum and maximum are 1.3 and 7.9, the resulting range of rounded numbers is 1.2 through 8.0, because 1.2 is the nearest multiple of 0.2 that is less than 1.3, and 8.0 is the nearest multiple of .2 that is greater than 7.9.

NOTE

> RGL/11 rounds only the minimum and maximum values that are displayed at the first and last cells. If you want rounded–number labels on the intervening cells, you must specify a number of cells (in your call to DPAPER) that is a multiple of 2, 5, or 10.

For logarithmic scales, RGL/11 calculates the minimum rounded number by finding the power of 10 that is lower than the lowest value in the given data. The maximum is found using the following equation:

maximum = minimum * 10^c

where c is the number of cycles on that axis

For example, if the minimum value was 6., the minimum rounded number would be 1. If the graph paper had been drawn with four cycles, the axis would be scaled from 1. to 10000. If the graph paper had three cycles, it would be scaled from 1. to 1000.

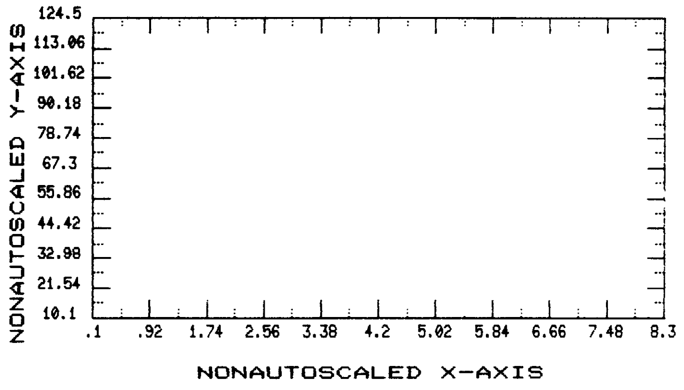
RGL/11 automatically generates rounded numbers and numeric labels whenever you select log paper, even if you supply a value of ".TRUE." in the "exact" argument of the LNAXIS subroutine.

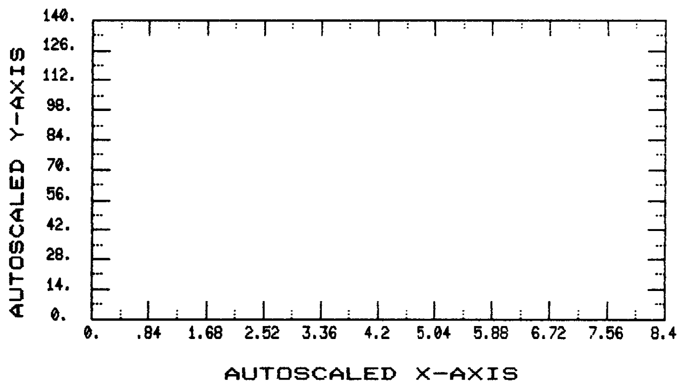
3.3.2 Autoscaling

Autoscaling is the procedure RGL/11 uses in certain circumstances. It establishes the range of values against which to plot your data (the numbers within the range can be either rounded numbers or exact numbers). RGL/11 derives this range from the given data. It can autoscale either the x–axis or y–axis or both. To invoke autoscaling, you default the "minvalue" and "maxvalue" arguments in the LNAXIS subroutine call, or use just the PDATA subroutine call to plot your data (see Section 3.1).

Figure 3–7 shows two pieces of graph paper: the top one uses the user–specified scale and exact values at the cell boundaries, the bottom one is autoscaled and uses rounded numbers at the cell boundaries.

Figure 3–7: Non-Autoscaled, Exact-Numbered Graph Paper and Autoscaled, Round-Numbered Graph Paper





3.3.3 Multi–Scaled Graphs

Multi–scaled graphs have one x–axis and multiple y–axis scales. Using this type of graph, you can plot two different sets of data against a common variable represented on the x–axis scale.

To create a multi–scaled graph, call the DPAPER subroutine, then the LNAXIS subroutine once to scale the x–axis, then call it twice again to scale the two y–axis options: y–left and y–right. Once you have established the grid paper, you can call the PDATA routine twice to plot the successive sets of data against the appropriate y–axis.

To identify the axis you want affected by the LNAXIS call, specify one of the following codes in its " 'axisid' " argument:

- XB (x–bottom)
- YL (y–left)
- XT (x–top)
- YR (y–right)

You can create two y–axis scales only if they both use the same x–axis scale.

After you have drawn and scaled the multi–scaled paper, you can instruct RGL/11 to plot data against the appropriate x– and y–scale combination by passing a " 'yaxis' " argument that is a single character representing the axes as follows:

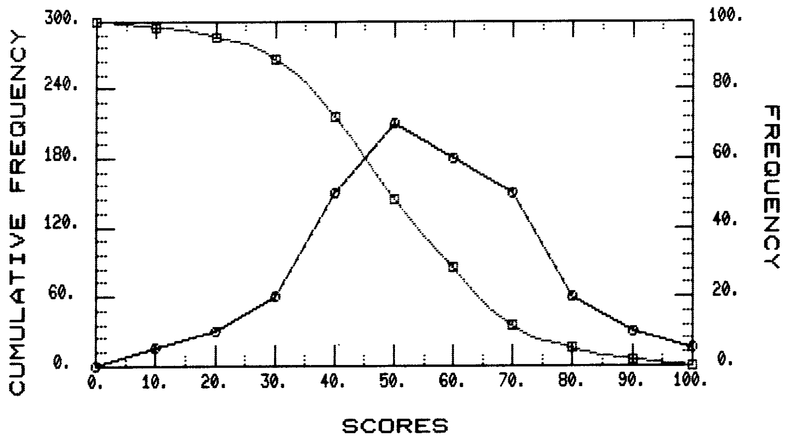
- L (y–left)
- R (y–right)

The following program, named MULTI on your distribution volume, plots two sets of data against a common x–axis scale and two y–axis scales. Figure 3–8 shows the plot generated by the example.

```
C   MULTI
C   This program plots the cumulative frequency distribution of a
C   set of hypothetical test scores against the left y-axis and
C   "the frequency distribution polygon" of the same data against
C   the right y-axis.
```

```
C
      DIMENSION SCORES (11), CUMFRQ (11), FREQ (11)
      DATA SCORES /0.,10.,20.,30.,40.,50.,60.,70.,80.,90.,100./
      DATA CUMFRQ /300.,295.,285.,265.,215.,145.,85.,35.,15.,5.,0./
      DATA FREQ /0.,5.,10.,20.,50.,70.,60.,50.,20.,10.,5./
C
      CALL INITGR (5)
      CALL CLRSCR
      CALL CLRTXT
C
C  Draw linear-by-linear grid with 10 cells and 2 subcells on
C  the x-axis and 5 cells and 5 subcells on the y-axis. Grid
C  color is GRAY2 and the axes are ungridded.
C
      CALL DPAPER ('LIN',10,2,'LIN',5,5,'GRAY2')
C
C  Scale bottom x-axis with values between 0 and 100 and, at each
C  of the boundaries, display a label "0", "10", and so forth,
C  using exact values.  Give the axis the label "SCORES."
C
      CALL LNAXIS ('XB','SCORES',0.,100.,.TRUE.)
C
C  Scale left y-axis with values between 0 and 300; label each
C  of the 5 cells (0., 60., 120., 180., 240., 300.), using
C  exact values; label the axis "CUMULATIVE FREQUENCY."
C
      CALL LNAXIS ('YL','CUMULATIVE FREQUENCY',0.,300.,
     1 .TRUE.)
C
C  Scale right y-axis with values between 0 and 100; label
C  each of the 5 cells (0., 20., 40., 60., 80., 100.), using
C  exact values; label the axis "FREQUENCY."
C
      CALL LNAXIS ('YR','FREQUENCY',0.0,100.0,.TRUE.)
C
C  Plot cumulative frequency distribution of eleven elements
C  from the SCORES array and the CUMFRQ array against the left
C  y-axis, writing the data line in GRAY1, using marker 1
C  (a dot), and linetype 1 (a solid line), with the smoothing
C  function enabled, shading disabled, and shadeline argument
C  defaulted.
C
      CALL PDATA (11,SCORES,CUMFRQ,'L','GRAY1',1,1,
     1 .TRUE.,.FALSE.,)
C
C  Now plot frequency distribution polygon of eleven elements
C  from SCORES and FREQ arrays against the right y-axis, writing in
C  GRAY2, using marker 2 (an octagon), linetype 1, with
C  smoothing and shading both off.
C
      CALL PDATA (11,SCORES,FREQ,'R','GRAY2',2,1,.FALSE.,
     1 .FALSE.,)
      END
```

Figure 3–8: Multi–Scaled Graph Plotted by the MULTI Program



3.3.4 Labeling the Cells of an Axis

The above example shows how the LNAXIS subroutine can scale and label the cells of an axis with numeric values and give the axis a title. Notice that the number of labels is one more than the number of cells. If there are four cells, for example, there are always five labels.

Another subroutine, LTAXIS, can also label the cells of an axis and assign numeric values to the cell boundaries, but LTAXIS can display either text or numeric labels, not just numeric labels, at the cell boundaries.

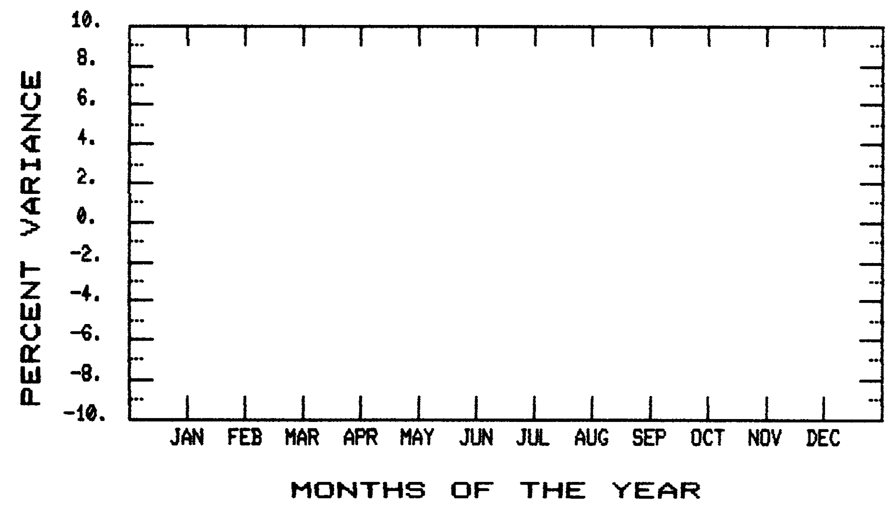
Both LNAXIS and LTAXIS write in overlay mode, the mode established by the DPAPER call. (DPAPER must be called before a LNAXIS or LTAXIS call.) Refer to Figure 3–1 to see the areas on a graph where the LNAXIS and LTAXIS labels are displayed.

The following program, named LABEL2 on your distribution volume, defines a grid and gives it both numeric and text labels. Figure 3–9 shows the plot generated by the program.

```
C  LABEL2
C  This program draws a grid with numeric and text labels.
C
         CALL INITGR (5)
         CALL CLRSCR
         CALL CLRTXT
C
C  Draw a grid with a blank linear x-axis having 13 cells and 1
C  subcell each (in effect, no subcell, since it is coextensive
C  with the cell itself) and a blank linear y-axis having 10
C  cells and 2 subcells each.  The lines will be drawn in GRAY3.
C
         CALL DPAPER ('LIN',13,1,'LIN',10,2,'GRAY3')
C
C  Scale the bottom x-axis with values between 0 and 13 and label
C  it "MONTHS OF THE YEAR." Label each of the 13 cells with
C  3-character labels (the first is blank, the second JAN, and so
C  forth).
```

```
C
      CALL LTAXIS ('XB', 'MONTHS OF THE YEAR', 0.0, 13.0,
      13,'   JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC')
C
C  Scale the left y-axis with values between -10 and 10 and label
C  it "PERCENT VARIANCE." Label each of the 10 cells with exact
C  values.
C
      CALL LNAXIS ('YL','PERCENT VARIANCE',-10.0,10.0,.TRUE.)
      END
```

Figure 3–9: Graph Plotted by the LABEL2 Program (A Labeled Graph)



3.4 Selecting Line Patterns and Markers

RGL/11 can connect points on a graph with nine different line patterns and mark them with 11 different markers. These are the same line patterns and markers as in the SLNPAT and MARKER subroutines of the picture–drawing subroutines. However, when you are plotting data, you can select the line pattern and marker you want by arguments in the PDATA subroutine, and you can select the marker by an argument in PPOINT.

The most commonly used line patterns are:

Dashed line (Pattern 9)	— This is the default line pattern when the shading option is in effect.
Solid line (Pattern 1)	— This is the initial default.
Blank line (Pattern 0)	— RGL/11 plots the coordinate pairs, but does not connect them (in effect, creating a "point plot").

The other line patterns are listed in the SLNPAT subroutine description in Chapter 5.

The marker codes are integers from 0 through 10 and from 100 through 110. Each number from 0 through 10 represents a different marker. The markers they represent are listed in the MARKER subroutine description in Chapter 5.

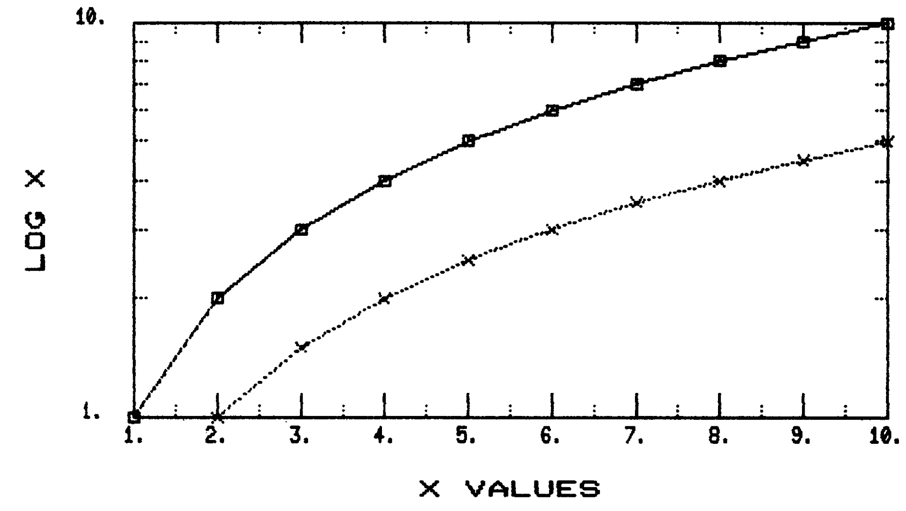
When you specify a number in the 0–10 range, RGL/11 places the marker at every point. The numbers from 100 through 110 represent the same eleven markers. However, when you specify a number in this range, RGL/11 places the marker at every tenth point.

If you default the marker argument, no marker is displayed.

The following program, named LINTYP on your distribution volume, plots two sets of data in different line patterns and marks the points with different markers. Figure 3–10 shows the data lines generated by the program example.

```
C   LINTYP
C   This program plots the data path of two equations:
C     Y = Log X            and            Y = Log X/2
C
          DIMENSION XDATA (10), YDATA1(10), YDATA2(10)
          DATA XDATA /1.,2.,3.,4.,5.,6.,7.,8.,9.,10./
          DATA YDATA1 /1.,2.,3.,4.,5.,6.,7.,8.,9.,10./
          DATA YDATA2 /.5,1.,1.5,2.,2.5,3.,3.5,4.,4.5,5./
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
C
C   Draw a grid with a blank linear x-axis having nine
C   cells and two subcells, and a blank single-cycle
C   logarithmic y-axis having nine subcells;  the grid
C   lines will be drawn in GRAY3.
C
          CALL DPAPER ('LIN',9,2,'LOG',1,9,'GRAY3')
C
C   Scale the bottom x-axis in exact values from 1. to 10.,
C   and label it "X VALUES." Autoscale the left y-axis and
C   label it "LOG X."
C
          CALL LNAXIS ('XB','X VALUES',1.,10.,.TRUE.)
          CALL LNAXIS ('YL','LOG X',1.,1.,)
C
C   Plot the ten elements in XDATA and YDATA1 against the
C   left y-axis, marking the data points with a square and
C   drawing a solid line colored GRAY3.  Smoothing and
C   shading are both disabled.
C
          CALL PDATA (10,XDATA,YDATA1,'L','GRAY3',1,1,.FALSE.,
          1.FALSE., )
C
C   Plot XDATA and YDATA2, using an X to mark the data points
C   and a dashed line for the plot.  Again both smoothing and
C   shading are disabled.
C
          CALL PDATA (10,XDATA,YDATA2,'L','GRAY1',5,4,.FALSE.,
          1.FALSE., )
          END
```

Figure 3–10: Graph with Alternate Line Patterns and Markers



3.5 Smoothing Data Curves

When you plot a set of coordinates, RGL/11 provides the option of connecting the points either with a series of straight lines or with a smooth curve. You select the type of line by supplying either ".TRUE." or ".FALSE." as the value for the "smooth" argument of the PDATA subroutine, the eighth PDATA argument. The value ".TRUE." gives the smoothed data line, ".FALSE." gives straight lines. The default is ".FALSE.", straight lines.

The algorithm that performs the smooth–curve interpolation is called a *spline–fitting algorithm.* It requires that the x–coordinates in the input array be passed in ascending order. If they are not, the data path will be erratic.

Sometimes smoothing a curve causes the new curve to "overshoot" and extend beyond the edges of the graph without being clipped. When this happens, one way to correct the problem is to "scale down" the data by changing the scale on the appropriate axis; that is, by changing the maxvalue and minvalue arguments of a LNAXIS call for that axis.

The following program, named SMOOTH on your distribution volume, illustrates the smoothing option. Figure 3–11 shows the data path generated by this program.

```
C   SMOOTH
C   This program plots the data path of the polynomial function:
C               Y = X[3] + 3X[2] - X - 3
C
        DIMENSION XARRAY (9), YARRAY (9)
        DATA XARRAY /-3.7,-3.5,-3.,-2.,-1.,0.,1.,1.5,1.6/
        DATA YARRAY /-7.,-5.63,0.,3.,0.,-3.,0.,5.63,7./
C
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
```

```
C
C   Draw a gridded linear x-axis with nine cells and no subcells.
C   ("1" subcell is coextensive with the cell.)  Draw a similar
C   y-axis with fourteen cells and no subcells.  The grid lines
C   are drawn in GRAY3.
C
          CALL DPAPER ('GLIN',9,1,'GLIN',14,1,'GRAY3')
C
C   Scale the bottom x-axis from -5. to 4.; use exact values at the
C   cell boundaries and supply an axis title.  Scale the left
C   y-axis from -7. to 7., and use rounded numbers at the cell
C   boundaries.
C
          CALL LNAXIS ('XB','Y = X-CUBED + 3(X-SQUARED) - X - 3 ',
          1-5.,4.,.TRUE.)
          CALL LNAXIS ('YL',' ',-7.,7.,.TRUE.)
C
C   Plot the nine elements in XARRAY and YARRAY with respect to
C   left y-axis.  Draw the data line and markers in GRAY3, marking
C   the points with a square and using a solid line.  Smoothing is
C   enabled, shading is disabled.
C
          CALL PDATA (9,XARRAY,YARRAY,'L','GRAY3',1,1,.TRUE.,
          1.FALSE.,)
          END
```

Figure 3–11: Smoothed Data Lines

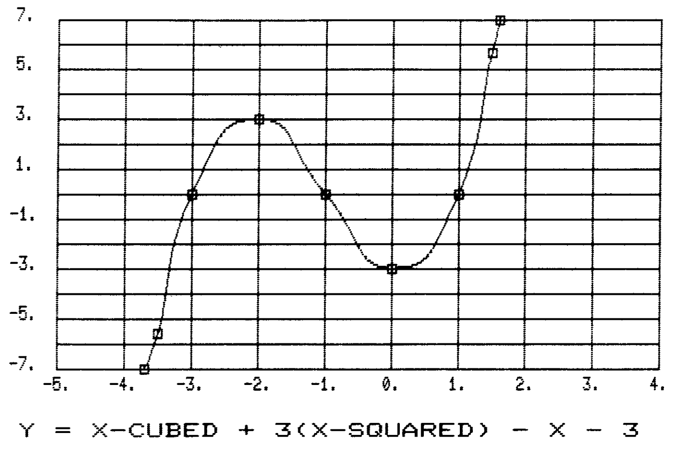
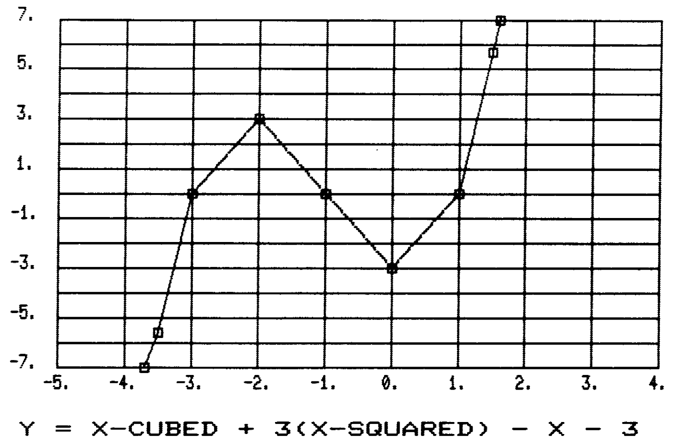


Figure 3–12 uses the same program, changing only the "smooth" argument in the PDATA call:

```
CALL PDATA(9,XARRAY,YARRAY,'L','GRAY3',1,1,.FALSE.,.FALSE.,)
```

Figure 3–12: Straight Data Lines



3.6 Creating Shaded Graphs

A *shaded graph* is a graph which is shaded from the apex of the data line downward to a horizontal shade line along the y–axis.

To turn shading on, you use the following arguments in the PDATA subroutine:

- Specify ".TRUE." in the shade argument, the ninth PDATA argument.
- Specify a shade line value in the tenth argument (yvalue) or use the default value which is the minimum value on the appropriate y–axis.
- Specify a linetype (for the shade pattern) in the seventh argument, or use the default value.

Each time a program calls PDATA with the shading enabled and a default linetype, the shading pattern changes. This change makes it easier to differentiate between shaded regions. PDATA has four special line patterns it uses before, on the fifth such call, it duplicates the first shade pattern.

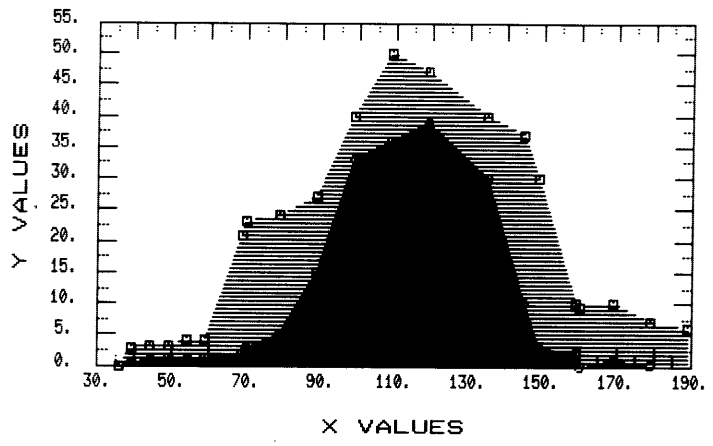
You do not have to use the default line patterns, however. They apply only if you do not specify a linetype.

The following program, named SHADE4 on your distribution volume, creates a shaded graph with two sets of data, two overlapping shaded areas. Figure 3–13 shows the plot generated by the program example.

```
C   SHADE4
C   This program creates a surface graph from two sets of
C   data; it shades the area under each curve in a
C   different shade pattern.
```

```
C
         DIMENSION XVALS (21), YVAL1 (21), YVAL2 (21)
         DATA XVALS /36.,39.,44.,49.,54.,59.,69.,70.,79.,89.,
         1 99.,109.,119.,135.,145.,149.,159.,160.,169.,179.,189./
         DATA YVAL1 /0.,3.,3.2,3.3,4.,4.,21.,23.,24.,27.,40.,50.,
         2 47.,40.,37.,30.,10.,9.4,10.2,7.,6./
         DATA YVAL2 /0.,.5,1.,1.2,1.,1.3,2.,3.,5.,15.,33.,36.,39.,
         3 30.,10.,3.,2.1,0.,1.,.1,-.2/
C
         CALL INITGR (5)
         CALL CLRSCR
         CALL CLRTXT
C
C   Draw a blank linear grid in GRAY3 with the x-axis having
C   sixteen cells, two subcells each, and the y-axis having
C   eleven cells, two subcells.
C
         CALL DPAPER ('LIN',16,2,'LIN',11,2,'GRAY3')
C
C   Scale the bottom x-axis with exact numbers from 30. to
C   190., and label the axis "X VALUES." Scale the left
C   y-axis with exact numbers from 0. to 55., and label the
C   axis "Y VALUES."
C
         CALL LNAXIS ('XB','X VALUES',30.,190.,.TRUE.)
         CALL LNAXIS ('YL','Y VALUES',0.,55.,.TRUE.)
C
C   Plot the 21 elements from the XVALS and YVAL1 arrays against the
C   left y-axis.  Draw in GRAY3, using a square marker and a
C   dashed line.  Smoothing is disabled and shading is enabled.
C   Shading extends to a horizontal line at point 0.0 on the left
C   y-axis.
C
         CALL PDATA (21,XVALS,YVAL1,'L','GRAY3',1,4,.FALSE.,.TRUE.,0.0)
C
C   Plot the second data set in YVAL2, changing the marker to an
C   octagon and the line pattern to a solid line.
C
         CALL PDATA (21,XVALS,YVAL2,'L',GRAY3',2,1,.FALSE.,.TRUE.,0.0)
         END
```

Figure 3–13: Shaded Graph with Two Sets of Data



3.7 Selecting Gray Shades or Color

To select gray shades or color for a graph, use the color arguments in the DPAPER subroutine, the PDATA subroutine, or both. The four possible colors are:

GRAY0	dark, the screen color
GRAY1	dark gray
GRAY2	light gray
GRAY3	white

On a color monitor, these colors are displayed as dark, blue, red, and green, respectively.

Typically, color number 0 (set to GRAY0) is the screen color and color number 1 (set to GRAY3) is the color for the grid lines, so there are two color numbers available for a program. Color numbers 2 and 3 alternate by default. They are set to GRAY2 and GRAY3 respectively by DPAPER. A program may use one color for data and another for the graph labels, for example, or two colors for data. It cannot use more than two colors for data. If it does, each color change after the second one changes the colors of points plotted two calls before it.

3.8 Plotting the Data

You can plot data from stored files or you can perform interactive data plotting.

3.8.1 Plotting Data from Stored Files

To plot data from stored files and exercise the many options RGL/11 makes available in creating graphs, you use the subroutines in the following sequence:

1. Using the DPAPER subroutine, you define the type of graph (linear or logarithmic) and type of axis (gridded or ungridded) that you want to use. (Section 3.2)
2. Using the LNAXIS subroutine, you assign numeric values to the cell divisions of a graph's axes; the values are printed at each cell boundary along the axis you specify. (Since the first cell is labeled at both its boundaries, there is one more label than there are cells along an axis.) LNAXIS also gives the entire axis the label you write into the " 'axislabel' " argument. (Section 3.3)
3. Using the LTAXIS subroutine, you assign text labels to the cell divisions of the top or bottom x–axis and left and right y–axes, and to the axis as a whole. (Section 3.3)
4. Using PDATA, you select the data arrays themselves, and also options such as shading and smoothing. You can use PDATA: (1) alone, (2) in combination with DPAPER so that you can have gridded and

logarithmic graphs, or (3) in combination with DPAPER and the labeling subroutines LNAXIS and/or LTAXIS. When you select PDATA argument values, they must agree with any values in preceding calls. (Section 3.4 to 3.7)

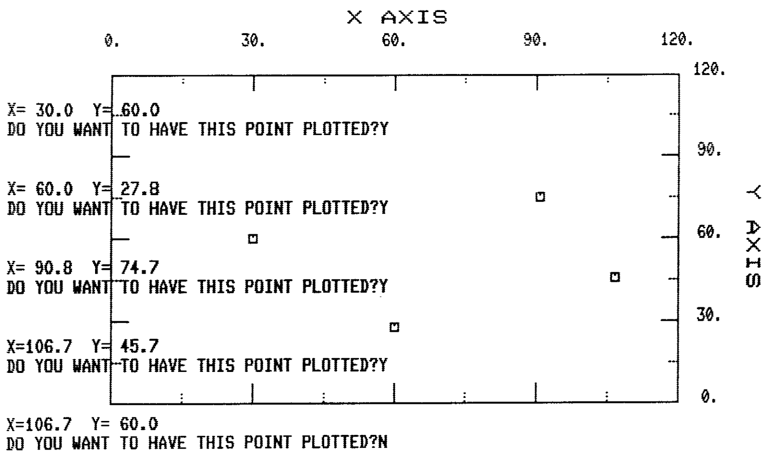
5. Using PPOINT, you can plot a single data point on a graph. (Section 3.8.2)

3.8.2 Interactive Data Plotting

To plot data interactively, you use the PPOINT subroutine. PPOINT enables you to plot single data points from the x– and y–coordinates you supply in the argument list. The y–coordinate can apply to either the left or right y–axis. You select the marker to mark the data point from the list of markers in the MARKER subroutine in Chapter 5; the default is 0, a dot.

An example, named POIN on your distribution volume, follows. In this example, you create a grid, label it, and call up the locator cursor. When you have moved the cursor to the location you want, you should exit from locator mode. You accomplish this by typing any key but the SHIFT, ESC, or DELETE key or arrow keys. After the exit from locator mode, the x and y values of the point at which you left the mode are printed on the screen. If they are what you want, you type an uppercase "Y" and PPOINT uses those values to plot the data point on the graph. The DO loop establishes five iterations of this procedure.

Figure 3–14: PPOINT Example



```
X= 30.0  Y= 60.0
DO YOU WANT TO HAVE THIS POINT PLOTTED?Y

X= 60.0  Y= 27.8
DO YOU WANT TO HAVE THIS POINT PLOTTED?Y

X= 90.8  Y= 74.7
DO YOU WANT TO HAVE THIS POINT PLOTTED?Y

X=106.7  Y= 45.7
DO YOU WANT TO HAVE THIS POINT PLOTTED?Y

X=106.7  Y= 60.0
DO YOU WANT TO HAVE THIS POINT PLOTTED?N
```

```
C   POIN
C   This program uses the locator cursor to establish
C   points on a graph; the PPOINT subroutine then
C   plots the points.
C
C   Establish data types and initialize the terminal.
C
        LOGICAL*1 RESPON
        REAL X,Y                    ! COORDINATES OF DATA POINT
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
```

```
C
C  Draw and label the graph.
C
       CALL DPAPER ('LIN',4,2,'LIN',4,2,'GRAY3')
       CALL LNAXIS('XT','X AXIS',0.,120.,.TRUE.)
       CALL LNAXIS('YR','Y AXIS',0.,120.,.TRUE.)
C
C  Use the locator cursor to mark points on a graph.
C  If they are the ones you want to plot, type an
C  uppercase "Y."
C
       DO 100 I=1,5
               CALL LFREE(X,Y,KEY,'R')
               TYPE 20,X,Y
20             FORMAT (' X=',F5.1,2X,'Y=',F5.1)
               TYPE 50
50             FORMAT (' DO YOU WANT TO HAVE THIS POINT PLOTTED?',$)
               ACCEPT 200,RESPON
200            FORMAT (A1)
               IF (RESPON.NE.'Y') GO TO 100
               CALL PPOINT(X,Y,'R',1)
100    CONTINUE
       END
```

The x– and y–coordinates for PPOINT can be supplied in other ways as well. For example, they can be typed from the keyboard or read from a file.

Chapter 4
Program Development

This chapter details the procedure for creating and running graphic programs. The procedure is called "program development." It consists of the following steps:

- Writing a graphic program, called the "source program"
- Using an editor to create a file, called a "source file", which contains the program, and then using the editor to edit the file as needed
- Using the FORTRAN compiler to convert the source file into an object module
- Using a linker or task–builder utility to generate an executable program
- Running the program

This chapter provides only an introduction to program development, and it contains information for creating and running graphic programs specifically for RT–11 and RSX–11M systems. Another source of information on program development is the *RSX–11M/M–PLUS Guide to Program Development* or the *RT–11 System User's Guide.*

NOTE

Sections 4.1 and 4.2 of this chapter describe writing and editing a source program. They concern both RT–11 and RSX–11M systems. Section 4.3 concerns compiling, linking, and running a program on an RT–11 system, and Section 4.4, compiling, task building, and running a program on an RSX–11M system.

4.1 Writing a Graphic Source Program

This section reviews the following RGL/11 conventions you should use in your graphic programs:

- Conventions for string expressions (Section 4.1.1)
- Initializing your program (Section 4.1.2)
- Program overlays (Section 4.1.3)

Other requirements for successful programs, such as the correct settings for your terminal, are in the sections on running the program (Section 4.3.3 for RT–11 systems and 4.4.3 for RSX–11M systems), in the *VT125 User's Guide*, and in the user's guide for your system.

NOTE

> Do not use VT125 macrographs in your graphic programs. The RGL/11 package uses all 26 macrographs. If macrographs also occur in user programs, they will overwrite the existing ones and cause indeterminate results.

4.1.1 String Expressions

When you pass a data string expression to a RGL/11 subroutine, its data type must be a LOGICAL*1 or byte data type, and the last element of the array must be a null (zero) byte.

When you pass a text string to a RGL/11 graphic text subroutine (either LINETX or TEXT) and you want to include a double quotation mark ("), you must precede it with another double quotation mark, for example:

```
TEXT ('Type ""START"" to begin program.')
```

The double quotation mark is a ReGIS command string terminator; therefore it requires this procedure in text strings of LINETX and TEXT subroutines, and in the axis labels of LNAXIS and LTAXIS.

4.1.2 Initializing Your Program

In order to set the initial conditions for your program to known values and to clear the screen, you should call these subroutines first:

- INITGR — sets the terminal

NOTE

> INITGR *must* be the first RGL/11 subroutine your program calls; if it is not, the program's performance will be unpredictable.

- CLRSCR — clears graphic images from the screen
- CLRTXT — clears regular text from the screen
- SWINDO — if you want to define your own coordinate system

Chapter 5 has the detailed descriptions of these subroutines, and Chapters 2 and 3 have some examples. The SWINDO concepts are described in Section 2.2, Defining Coordinate Systems.

4.1.3 Program Overlays

RGL/11 programs can be run in one of two ways: overlaid or non–overlaid. An overlaid program consists of two or more program segments. When an overlaid program is executed, only one program segment is in low memory at any one time.

Overlays provide a way to execute programs that would otherwise exceed the size of memory while sacrificing only some execution speed. Using overlays, you can build programs that are larger than the 32K word limit imposed by the PDP–11 architecture. But since overlays involve disk I/O, the execution speed of an overlaid program may be slower than it would be if not overlaid.

One type of RGL/11 application program that usually must be overlaid because of size is one that calls many RGL/11 data–plotting subroutines. A program that calls only RGL/11 picture–drawing subroutines probably does not need to be overlaid.

On RT–11 systems, there are two kinds of overlays: disk resident and extended memory resident. If the program will run under an SJ or FB monitor, the segments are stored on disk; if it will run under an XM monitor, the segments can be stored either on disk or in extended memory. Extended–memory overlays do not involve disk I/O, and therefore they are faster than disk–resident overlays. Another advantage of programs with extended–memory overlays is that they are run as virtual jobs, and therefore they have access to all 32K of memory.

For further information on overlays, see the *PDP–11 FORTRAN Language Reference Manual*, the *RT–11 System User's Guide*, or the *RSX–11M/M–PLUS Task Builder Manual*. Information on whether to use disk–resident or extended–memory–resident overlays (for RT–11 users) is in Sections 4.3.2.2 and 4.3.2.3 of this manual.

4.2 Creating and Editing the Source File

When you have created the source program, use a text editor to enter the program into a file. The file is the "source file" for your program.

As you type your program, be sure the program lines do not exceed 72 characters in length. The compiler, unless otherwise instructed, does not process characters beyond position 72. If the lines are longer than 72 characters, the compiler might print confusing error messages or cause your program to execute incorrectly even though it compiled correctly. On a VT125 terminal, you can enable the margin bell to sound when you reach position 72. See SETUP mode in the *VT125 User's Guide* for instructions.

4.3 Program Development on an RT–11 System

This section describes the steps for compiling, linking, and running a program on an RT–11 system.

4.3.1 Compiling the Program (RT–11)

"Compiling" is the process by which your FORTRAN compiler attempts to translate your source file into an object program (which is a program consisting of machine instructions). That procedure, made specific for graphic programs, is summarized below. If the compiler prints error messages, refer to the *RT–11, RSTS/E FORTRAN IV User's Guide* or the *RT–11 System Message Manual.*

There are two ways to compile your graphic program. You can choose to use:

1. The FORTRAN command
2. The indirect command file RGLLNK, which is especially written to compile, link, and run graphic programs

If you choose to use the FORTRAN command, type:

```
.FORTRAN prog RET
```

where:

prog	is the name of your program

If you choose to use the RGLLNK command file, it asks you questions and then uses your answers for the program development process you selected. (You can run RGLLNK from any terminal, but to execute a graphic program, you must be using a VT125 terminal.) To invoke RGLLNK, type:

```
.@RGLLNK RET
```

The RGLLNK questions are:

```
Program name [P] ?
Do you want to compile [Y] ?
Do you want listings [N] ?
What device do you want the listings to go to [LP:] ?
Do you want to link [Y] ?
Do you want a map [N] ?
What device do you want the listings to go to [LP:] ?
Do you want overlays [Y] ?
```

Do you want extended memory overlays [N] ?
Do you want to run [Y] ?

RGLLNK is treated in detail in Section 4.3.2.2.

4.3.2 Linking the Program (RT–11)

When you have successfully compiled your program so that you have it as an object file, the next and final step in generating an executable program is to link the object file the compiler created with the object files of libraries or other subprograms. The linker creates the executable program, which is called a load module, and in addition links your program to any subprograms your program calls.

For RGL/11 programs, you need to link your object file with:

1. One of the three RGL/11 libraries: RGLLIB.OBJ, PRMLIB.OBJ, or REXLIB.OBJ
2. BLODAT, the block data file

The decision about which RGL/11 library to use is dependent upon whether you want to use non–overlaid programs, overlaid programs (disk–resident or extended–memory–resident), or programs that use the alternate extended–memory overlay structure. When you have made that decision and linked your program accordingly, your program is linked to the correct library.

You may also want to link your object files with:

1. Your own library of subroutines
2. Other object files, libraries, or subprograms you name individually in the command line

There are three ways you can link a graphic program. You can:

- Use the RGLLNK command file, which you can use for any graphic program (described above and in detail in Section 4.3.2.2)
- Use the LINK command, which you should use only for a program that is not overlaid (Section 4.3.2.1)
- Edit and then run the command file RGLOVR.COM, which you should use only for programs that will use the alternate extended–memory overlays and need additional RGL/11 routines overlaid and, possibly, some additional user routines. (Section 4.3.2.3)

4.3.2.1 The LINK Command —

Use the LINK command only for a graphic program that you do not intend to overlay. To use it, type:

```
.R LINK (RET)
*myprog = myprog,BLODAT,RGLLIB/B:1200/P:1000 [,filespec](RET)
```

where:

myprog	is the name of your program
BLODAT	is the name of the object file that defines the initial graphic defaults
RGLLIB	is the name of the graphics library for non–overlaid programs
/B:1200	ensures there will be enough room in the monitor stack
/P:1000	allows table space for the linker program
filespec	is the name or names (separated by commas)for optional subroutines, or for optional libraries

If the linker encounters no fatal errors, it produces an executable program called myprog.SAV.

Like the LINK command, the RGLLNK command file can also produce a non–overlaid program, so you have a choice which procedure to use. RGLLNK has the advantage of automatically linking your program with the correct RGL/11 library.

4.3.2.2 The Indirect Command File, RGLLNK —

RGLLNK creates and executes a command file called RGL.COM that enables you to compile, link, and execute your program. If you are running under the XM monitor and you want to use overlays, you have the option of using low memory overlays or extended–memory overlays. If you choose extended memory overlays, your program will be created as a virtual job. If you are running under FB or SJ and want to use overlays, you can use only low memory overlays. You can default all answers to RGLLNK except the program name. The default for each question is given in square brackets at the end of each question.

To invoke RGLLNK, type:

```
.@RGLLNK(RET)
```

A sample RGLLNK dialog follows. It compiles, links, and runs a program named prog:

```
.@RGLLNK(RET)
.RUN RGLLNK
Program name [P]? prog(RET)
Do you want to compile [Y] ? (RET)
Do you want listings [N] ? (RET)
```

Type Y(RET) if you want listings and (RET) if you do not. If you answer Yes, RGLLNK prompts with the further question:

```
What device do you want the listings to go to [LP:] ?
```

Type RET to have the listing go to the line printer, or type the name of the device, such as DL0:, where you want the listings sent.

```
Do you want to link [Y] ? RET
Do you want a map [N] ? RET
```

Type YRET if you want a load map; type RET if you do not. If you answer Yes, RGLLNK prompts with the further question:

```
What device do you want the map to go to [LP:] ?
```

Type RET to have the load map go to the line printer, or type the name of the device, such as DL0:, where you want it sent.

```
Do you want overlays [Y] ? RET
```

Type RET to overlay your programs; type NRET to avoid overlays. A Yes answer links your program to PRMLIB.OBJ., and RGLLNK prompts with the further question:

```
Do you want extended memory overlays [N] ? RET
```

Type YRET if you are running under XM and want extended–memory overlays, otherwise type RET.

```
Do you want to run [Y] ? RET
```

Type RET if you have a VT125 and want to run the program; type NRET if you do not.

If you are running under SJ, FB, or XM and want to compile, link, and run a program called PROG1 using low memory overlays, RGL.COM, the file RGLLNK creates, would look like this:

```
FORT/CODE:THR PROG1
R LINK
PROG1=PROG1,BLODAT,PRMLIB/B:1200/A/W/P:1000//
LFIXED,LOCAT2/O:1
LFREE,LOCATE/O:1
PPOINT/O:1
PDATA/O:1
DPAPER/O:1
LNAXIS/O:1
LTAXIS/O:1
LNNICE,LINMIN,LINMAX/O:2
PRINUM,PRISTR,TEXT,XLABEL,YLABEL/O:2
FMINMX,LGNICE,EXPTST/O:2
DPALOG,DPALIN,DEFALT,DRWBOX/O:2
LINE,MARKER,MOVE,SWINDO/O:3
GENOVR/O:3
INITGR,CLIPIT,SSTATE/O:3//
^C
RUN PROG1
```

If you are running under the XM monitor and want to compile, link, and run PROG1 using extended–memory overlays, RGL.COM looks like this:

```
FORT/CODE:THR PROG1
R LINK
SY:PROG1=PROG1,BLODAT,PRMLIB/I/A/W/P:1000//
LFIXED,LOCAT2/V:1
LFREE,LOCATE/V:1
PPOINT/V:1
PDATA/V:1
DPAPER/V:1
LNAXIS/V:1
LTAXIS/V:1
LNNICE,LINMIN,LINMAX/V:2
PRINUM,PRISTR,TEXT,XLABEL,YLABEL/V:2
FMINMX,LGNICE,EXPTST/V:2
DPALOG,DPALIN,DEFALT,DRWBOX/V:2
LINE,MARKER,MOVE,SWINDO/V:3
GENOVR/V:3
INITGR,CLIPIT,SSTATE/V:3//
$QBLK

^C
R PROG1
```

4.3.2.3 The Indirect Command File, RGLOVR —

Using the indirect command file RGLOVR is the third way to link a program. Use it for programs that use extended–memory overlays. It provides an alternate extended–memory overlay structure. This overlay structure will overlay additional RGL/11 subroutines. You should use RGLOVR only if you are running under XM, are going to use extended–memory overlays, and:

- Your program is too large for the extended–memory overlay structure provided by RGLLNK and you would like to overlay more RGL/11 subroutines, or
- Your program is too large and you would like to overlay more RGL/11 subroutines and add your own virtual overlay segments to overlay region 4.

The contents of RGLOVR.COM are:

```
R LINK
SY:MAINPR=MAINPR,BLODAT,REXLIB/I/A/W/P:1000//
LFIXED,LOCAT2/V:1
LFREE,LOCATE/V:1
PPOINT/V:1
PDATA/V:1
DPAPER/V:1
LNAXIS/V:1
LTAXIS/V:1
ARCOVR/V:1
RELOVR/V:1
SAVOVR/V:1
GN2OVR/V:1
LNNICE,LINMIN,LINMAX/V:2
PRINUM,PRISTR,TEXT,XLABEL,YLABEL/V:2
```

```
FMINMX,LGNICE,EXPTST/V:2
DPALOG,DPALIN,DEFALT,DRWBOX/V:2
LINE,MARKER,MOVE,SWINDO/V:3
GENOVR/V:3
INITGR,CLIPIT,SSTATE/V:3//
$QBLK

^C
```

If you want to just overlay more RGL/11 subroutines, do the following:

1. Make a copy of RGLOVR.COM
2. Edit the copy by replacing all occurrences of "MAINPR" with the name of your own program

After the copy has been edited, it is ready to be run.

If you want to overlay more RGL/11 subroutines and overlay parts of your application program, do the following:

1. Make a copy of RGLOVR.COM
2. Edit the copy to:
   - Replace all occurrences of MAINPR with the name of your program
   - Include your segments in virtual overlay region 4. Remember to remove the "//" at the end of line 21 of RGLOVR.COM ("INITGR,CLIPIT,SSTATE/V:3//") and place it after your last user segment. If you choose to do this, you reduce the address space of the root by 4K (see the *RT System User's Guide, Extended Memory Overlays*).

An example of a RGLOVR command file containing user overlay segments follows. The example file is an edited copy of RGLOVR named PROG1.COM. The name PROG1 replaces MAINPR throughout, and two user overlay segments are added to region 4.

```
R LINK
SY:PROG1=PROG1,BLODAT,REXLIB/I/A/W/P:1000//
LFIXED,LOCAT2/V:1
LFREE,LOCATE/V:1
PPOINT/V:1
PDATA/V:1
DPAPER/V:1
LNAXIS/V:1
LTAXIS/V:1
ARCOVR/V:1
RELOVR/V:1
SAVOVR/V:1
GN2OVR/V:1
LNNICE,LINMIN,LINMAX/V:2
PRINUM,PRISTR,TEXT,XLABEL,YLABEL/V:2
FMINMX,LGNICE,EXPTST/V:2
DPALOG,DPALIN,DEFALT,DRWBOX/V:2
LINE,MARKER,MOVE,SWINDO/V:3
GENOVR/V:3
```

```
INITGR,CLIPIT,SSTATE/V:3
USER1/V:4
USER2/V:4//
$QBLK

^C
```

Execute RGLOVR (the edited copy of RGLOVR named PROG1) by typing:

```
.@PROG1 (RET)
```

When PROG1 successfully completes, it produces on the system device an executable program or load module named PROG1 with a .SAV extension.

RGL/11 extended–memory overlays are run as virtual jobs and use the R command, rather than the RUN command. The R command can load programs only from the system disk. To run a file named PROG1.SAV that was created by PROG1.COM, you would type:

```
.R PROG1 (RET)
```

4.3.3 Running the Program (RT–11)

If you are running under the FB or XM monitor or SJ with multi–terminal support, you must set your terminal to prevent RT–11 from sending carriage–return/line–feed characters. To do so, type:

```
.SET TT NOCRLF (RET)
```

To avoid typing this command every time you use RGL/11, add it to your startup indirect command file, which is either STARTS.COM, STARTF.COM, or STARTX.COM.

To run most RGL/11 programs, type:

```
.RUN prog (RET)
```

where:

prog   is the name of your program

If your program uses RGL/11 extended–memory overlays, however, it must be run from the system disk. The reason for this is that RGL/11 creates the program as a virtual job and RT–11 looks on the system disk for virtual jobs. The command line to run a virtual job uses the R command:

```
.R prog (RET)
```

where:

prog   is the name of the program using RGL/11 extended–memory overlays

When you issue either form of the RUN command, RT–11 attempts to execute the load module, prog.SAV. If it encounters a condition that prevents it from doing so, it prints a FORTRAN OTS (Object Time System) error message. For help in understanding the error message, see the *RT–11, RSTS/E FORTRAN IV User's Guide.*

4.4 Program Development on an RSX–11M System

This section describes the steps for compiling, task building, and running a program on an RSX–11M system.

4.4.1 Compiling the Program (RSX–11M)

"Compiling" is the process by which your FORTRAN compiler attempts to translate your source file into an object program (which is a program consisting of machine instructions). That procedure, made specific for graphic programs, is summarized below. If the compiler prints error messages, refer to Appendix C of the *PDP–11 FORTRAN 77 User's Guide.*

NOTE

The commands in this chapter give both MCR and DCL commands when these commands differ. All RGL/11 command files can handle both DCL and MCR commands.

There are two ways to compile your graphic program. You can use:

1. The FORTRAN compiler command. To do so, type:

   in MCR mode: `>F77 prog=prog (RET)`
   in DCL mode: `>FORT/F77 prog (RET)`

   where:

   prog   is the name of your program

2. The RGLLNK.CMD, an indirect command file that enables you to compile, link, and run your program. Since RGLLNK was copied to the system disk when RGL/11 was installed, you invoke it by typing:

   ```
   >@LB:[1,202]RGLLNK (RET)
   ```

   RGLLNK is copied into area LB:[1,202]. You may want to copy it into your own area for convenience. You can run RGLLNK from any terminal; however, to run the graphic program it builds you must have a VT125. RGLLNK is treated in detail in Section 4.4.2.2.

4.4.2 Task Building the Program (RSX–11M)

When you have successfully compiled your program so that you have it as an object file, the next and final step in generating an executable program is to link the object file the compiler created with the object files of libraries or other subprograms. The task builder creates the executable program, which is called a task image, and in addition links your program to any subprograms your program calls.

For RGL/11 programs, you need to link your object file with:

- The RGL/11 library: RGLLIB.OLB
- The block data file: BLODAT

Besides the RGLLNK.CMD command file, which can be used for any graphic program, overlaid or not (Section 4.4.2.2), there are two other ways you can task build a graphic program. You can:

- Run the task builder command, TKB (if your terminal is in DCL mode, the command is LINK), for programs that are not overlaid (Section 4.4.2.1)
- Run the RGLOVR.CMD indirect command file for programs that are overlaid (Section 4.4.2.3)

4.4.2.1 The Task Build Command —

The task builder command line for non–overlaid graphic programs is:

in MCR mode:

```
>TKB prog/FP=prog,LB:[1,202]RGLLIB/LB:BLODAT,LB:[1,202]RGLLIB/LB[,filespecs] RET
```

where:

/FP	indicates that your system has a floating point unit
prog	is the name of your program
BLODAT	is the "block data" file that sets initial defaults for all variables in the common area; you must explicitly reference it
RGLLIB/LB	is the library for non–overlaid RGL/11 programs
filespecs	is the name (or names, separated by commas) of any subroutine or subprogram you want to include (use the /LB switch if it is a library)

In DCL mode:

```
>LINK/CODE:FPP prog,LB:[1,202]RGLLIB/LIB/INC:BLODAT,LB:[1,202]RGLLIB/LIB[,filespecs] RET
```

4.4.2.2 The Indirect Command File, RGLLNK.CMD —

To invoke RGLLNK, type:

```
>@LB:[1,202]RGLLNK RET
```

You may use this command line whether your terminal is set for MCR or DCL commands. RGLLNK sets the CLI (Command Language Interpreter) to the appropriate setting. To avoid having to type the LB:[1,202] UIC for RGLLNK each time you use it, you may prefer to copy RGLLNK to your area.

The following example compiles a program called MYPROG.FTN and task builds it with RGLLIB:

```
>@LB:[1,202]RGLLNK(RET)
;RGLLNK.CMD AUTOMATIC COMPILE AND LINK TO RGL/11 LIBRARY
>SET TERM MCR
>* Enter file name of program [S]: MYPROG(RET)
>* Do you want to compile [Y/N D:Y]: (RET)
>* Do you want listings [Y/N D:N] (RET)
>* Do you want to task build? [Y/N D:Y]: (RET)
>* Do you want overlays? [Y/N D:Y] (RET)
>* Do you want a map [Y/N D:N] (RET)
>* Do you want to run ? [Y/N D:Y]: (RET)
```

If you answer Yes to the questions about listings and a map, RGLLNK asks these additional questions:

```
>* Do you want the listings to go to the line printer? [Y/N D:N]:
>* Do you want the map to go to the line printer? [Y/N D:N]:
```

If you answer No to the question, a file is created with a file name of MYPROG on the default device and default UIC. The listings file has a default of .LST and the map file has a default of .MAP.

4.4.2.3 The Indirect Command File, RGLOVR.CMD —

A third way to task build your program is to use RGLOVR.CMD to generate a task image from an overlaid program. RGLOVR.CMD provides a way to link your program using overlays without having to run RGLLNK. RGLOVR.CMD uses the RGLOVR.ODL file, which contains the overlay description used by the task builder. (Print out RGLOVR.ODL if you want to see the RGL/11 overlay structure.) To use RGLOVR.CMD, replace all occurrences of MAINPR with the name of your source file in both RGLOVR.CMD and RGLOVR.ODL.

To use RGLOVR, you must copy RGLOVR.CMD and RGLOVR.ODL to your own area, not leaving them in LB:[1,202].

4.4.3 Running the Program (RSX–11M)

When you successfully task build your program, you have a runnable program. You must run your graphic programs on a VT125, but before running them, be sure the terminal driver is set to prevent the operating system from sending carriage–return/line–feed characters. These characters interfere with the operations of the RGL/11 subroutines. To set your terminal to prevent RSX–11M from sending carriage–return/line–feed characters to the terminal, type:

for MCR mode	`>SET /NOWRAP=TI: RET`
for DCL mode	`>SET TERM NOWRAP RET`

Add this command to the system STARTUP.CMD file or to your LOGIN.CMD file to avoid typing it each time you use RGL/11.

Before you run your program, be sure that RGL/11's ERRTXT.TXT file is in your default area. Having it there will insure that RGL/11 error messages can be displayed on your screen. If it is not in your area, copy it from LB:[1,202].

To run the program, type:

```
>RUN prog RET
```

RSX–11M attempts to run the task image, prog.TSK. If it encounters a condition that prevents it from executing the program, it prints a FORTRAN Object Time System (F4POTS) error message. For help in understanding the error message, see Appendix C of the *PDP–11 FORTRAN 77 User's Guide.*

Chapter 5
RGL/11 Subroutines

This chapter contains descriptions of all RGL/11 subroutines. The subroutines are listed in alphabetical order according to their 6–character mnemonic name. Where necessary for clarity, the mnemonic is expanded. Each subroutine starts on a new page so that you can easily skim to find the one you want.

The subroutine descriptions contain the following paragraphs:

PURPOSE

explains what the subroutine does and how it affects the position of the graphic cursor. If the subroutine has an initial default, it is listed here.

FORM

shows the format of the subroutine statement. All its arguments are listed in order. Optional arguments are enclosed in braces ({ }). To call the subroutine, you use the CALL command with the subroutine statement as its argument. For example, the format for the CIRCLE subroutine is:

```
CIRCLE (radius)
```

A sample call to the CIRCLE subroutine is:

```
CALL CIRCLE (50.0)
```

ARGUMENT DESCRIPTION

gives each argument's name, data type, and effect. Data types are floating point, integer, or string. Arguments enclosed in single quotes indicate a string data type; the argument requires either a literal string or a variable that contains ASCII strings.

If the argument can be either a constant or a variable, it is called an *expression*. You can use either a string literal or a string variable in a *string expression*. A string expression must be a LOGICAL*1 data type (or array if longer than one character) that ends in a null (zero) byte, or a byte data type that is the default string data type for your RT–11 or RSX–11M system. A text string can pass a double quotation mark (") within the text only when two consecutive double quotation marks are used (" "). To display one, you must pass two.

Optional arguments are enclosed in braces ({ }). If you do not supply a value for an optional argument, the subroutine assumes a default, (see Section 4.1.2, Default Settings).

NOTE

Even when you do not supply a value for an optional argument, you must still include the punctuation that delimits it, for example:

```
MARKER ( , 10.0, 20.0)
```

In this example, the first argument is omitted and MARKER will use 0, a dot, as the default.

EXAMPLE
shows how to call the subroutine with sample arguments, and describes what the call does. If the subroutine takes no argument, it is simply named here.

RELATED SUBROUTINES
lists the subroutines that affect this subroutine or that are affected by it, that perform a similar function, or that perform the opposite function.

RESTRICTIONS
gives the limitations of the subroutine.

ERROR MESSAGES
lists any RGL/11 error messages that the subroutine can generate, suggests possible causes, and describes the circumstances of the error.

PROGRAM EXAMPLE
shows a program or program fragment that uses the subroutine in context.

PURPOSE

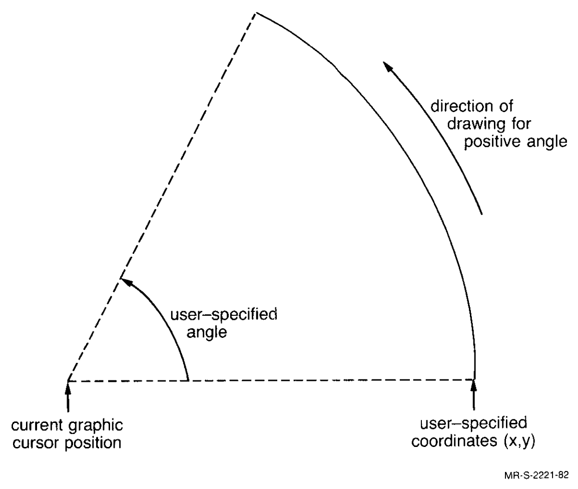
ARC draws an arc beginning at the location specified by the x and y arguments and ending at a point determined by the specified angle. The arc's center is at the current graphic cursor location.

The angle subtended by the arc has one side that is the line from the current graphic cursor location to the x,y coordinates you specify; the other side of the angle is the line drawn from the current graphic cursor location to the end of the arc, the arc being drawn until that line will form an angle of the specified size.

ARC draws in a counterclockwise direction when the angle is a positive number, and clockwise when it is negative.

After the arc is drawn, the graphic cursor is again at its starting location, the center of the arc. Figure 5–1 illustrates the ARC call.

Figure 5–1: Illustration of the ARC Subroutine


direction of drawing for positive angle
user–specified angle
current graphic cursor position
user–specified coordinates (x,y)
MR-S-2221-82

FORM

ARC (angle, x, y)

ARGUMENT DESCRIPTION

angle	is a floating point expression that defines the length of the arc. The unit of measure can be radians (default) or degrees. The sign of the angle affects the position of the angle, as being above or below the radius.
x and y	are floating point expressions that define the starting location of the arc in world coordinates.

EXAMPLE

```
ARC (2.29, 400.0, 300.0)
```

draws an arc whose starting location is at (400.0,300.0) and whose center is at the current graphic cursor location. The arc spans a length of 2.29 radians (about 135 degrees). After the arc is drawn, the graphic cursor is left at the center of the arc.

RELATED SUBROUTINES

SDGREE and SRADNS determine whether the angle argument is interpreted as degrees or radians (SRADNS is the default).

SLNPAT determines the line pattern that draws the arc (a solid line is the default).

SSHADE creates shading for the arc.

ARCC draws an arc beginning at the current graphic cursor location, rather than at the location you specify, and so the angle you define occurs at the current graphic cursor location, rather than at the specified location.

CIRCC, CIRCLE, and CIRCXY draw circles.

RESTRICTIONS

none

ERROR MESSAGES

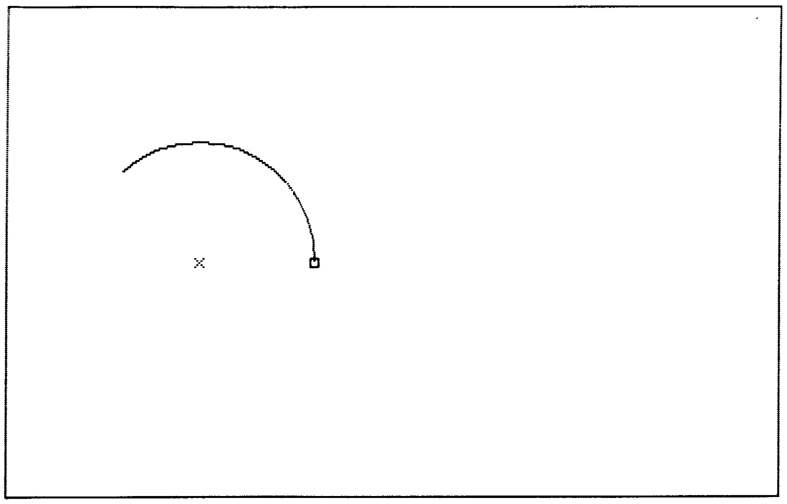
none

PROGRAM EXAMPLE

The following program initializes the terminal and draws a box to frame the screen. It then draws an arc starting at location (400.,300.) with the current graphic cursor location (250.,300.) as the center of the circle of which the arc is a part. The distance between the starting location and the current graphic cursor location is the radius of the circle; the greater the distance, the larger the circle. The length of the arc is 2.29 radians. See Figure 5–2.

```
C  Initialize VT125
      CALL INITGR(5)
      CALL CLRSCR
      CALL CLRTXT
      CALL SWINDO(0.,0.,1000.,625.)
C
C  Place a box around the whole screen
      CALL BOX(0.,0.,1000.,625.)
C
      CALL MOVE(250.,300.)
C
C  Place an 'X' at the current cursor location
      CALL MARKER(5,,)
C
C  Draw an arc
      CALL ARC(2.29,400.,300.)
C
C  Now put a box at the x and y coordinates
C  specified in the call to ARC
      CALL MARKER(1,400.,300.)
      END
```

Figure 5–2: Picture Illustrating an ARC Call



Another example is given in Section 2.1, General Strategy for Drawing Graphic Objects.

ARCC

PURPOSE

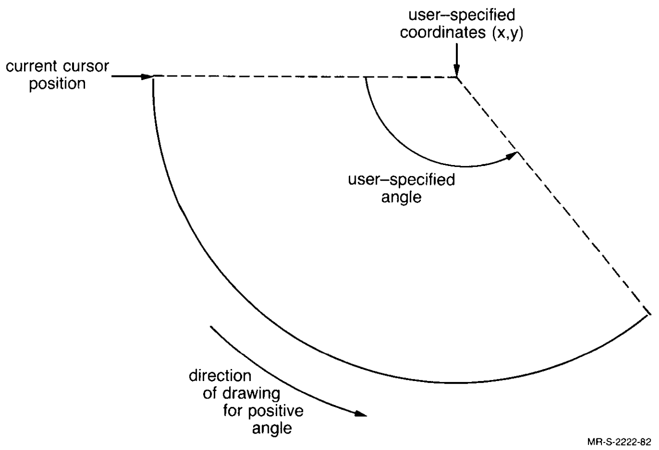
ARCC draws an arc beginning at the current graphic cursor location and ending at a point determined by the specified angle. The arc's center is at the coordinates you specify.

One side of the angle is the line from the location you specify to the current graphic cursor location. The other side of the angle is the line drawn from the location you specify to the end of the arc, the arc being drawn until the angle between the lines is the specified size.

ARCC draws in a counterclockwise direction when the angle is a positive number, and in a clockwise direction when it is negative.

ARCC leaves the graphic cursor location at the end of the arc. See Figure 5–3.

Figure 5–3: Illustration of the ARCC Subroutine



FORM

ARCC (angle, x, y)

ARGUMENT DESCRIPTION

angle	is a floating point expression that defines the length of the arc. The unit of measure can be radians (default) or degrees. If angle is a positive number, ARCC draws in a counterclockwise direction; otherwise, it draws clockwise.
x and y	are floating point expressions that define the center of the arc in world coordinates.

EXAMPLE

```
ARCC (2.29, 400.0, 300.0)
```

draws an arc that starts at the current graphic cursor location and spans 2.29 radians. The center of the arc is at location (400.0,300.0). After the arc is drawn, the graphic cursor is left at the end of the arc.

RELATED SUBROUTINES

SDGREE and SRADNS determine whether the angle argument is interpreted as degrees or as radians; SRADNS is the default.

SLNPAT determines the line pattern.

SSHADE creates shading for the arc.

ARC draws an arc beginning at the location you specify, rather than at the current graphic cursor location, and so the angle you define occurs at that location rather than at the current graphic cursor location.

CIRCC, CIRCLE, and CIRCXY draw circles.

RESTRICTIONS

none

ERROR MESSAGES

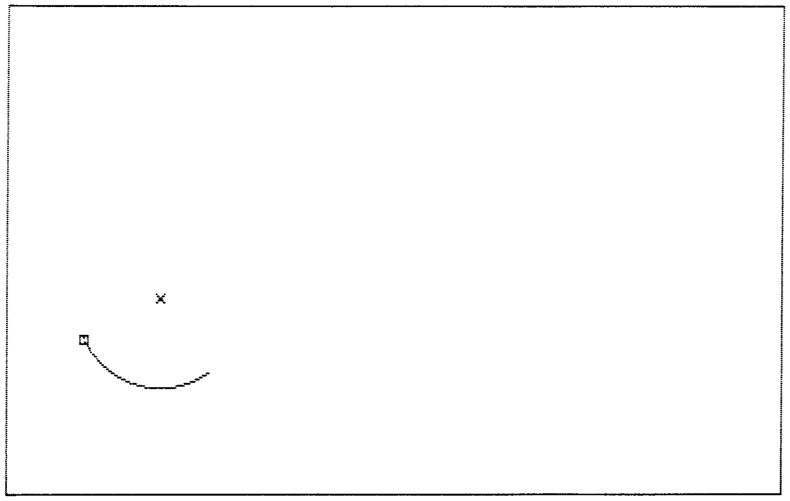
none

PROGRAM EXAMPLE

The following program draws an arc of 1.7 radians, starting at location (100.,200.), and having its center at location (200.,250.). See Figure 5–4.

```
C  Initialize VT125
      CALL INITGR(5)
      CALL CLRSCR
      CALL CLRTXT
      CALL SWINDO(0.,0.,1000.,625.)
C
C  Place a box around the whole screen
      CALL BOX(0.,0.,1000.,625.)
C
      CALL MOVE(100.,200.)
C
C  Place a box at the current cursor location
      CALL MARKER(1,,)
C
C  Now call ARCC.
      CALL ARCC(1.7,200.,250.)
C
C  Now put an 'X' at the x and y coordinates
C  specified in the call to ARCC
      CALL MARKER(5,200.,250.)
      END
```

Figure 5–4: Picture Illustrating an ARCC Call



Another example is given in Section 2.1, General Strategy for Drawing Graphic Objects.

PURPOSE

BOX draws a rectangle, defined by the two pairs of coordinates you supply to locate diagonally opposite corners of the box.

After the box is drawn, the graphic cursor is again at the first corner.

FORM

BOX (x1, y1, x2, y2)

ARGUMENT DESCRIPTION

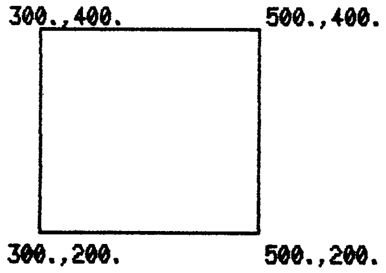
x1, y1, x2, and y2	are floating point expressions that define, in world coordinates, the coordinates for two diagonally opposite corners of the box.

EXAMPLE

```
BOX (300.,200.,500.,400.)
```

draws a box whose corners are defined by the coordinate pairs (300.,200.), (500.,200.), (500.,400.), and (300.,400.). The graphic cursor is left at location (300.,200.)

Figure 5–5: Picture Illustrating the BOX Call



RELATED SUBROUTINES

SLNPAT determines the line pattern.

SSHADE creates shading for the figure.

RELBOX also draws a box, but its arguments are the width and height of the box rather than the coordinates of two opposing corners.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

An example is given in Section 2.1, General Strategy for Drawing Graphic Objects.

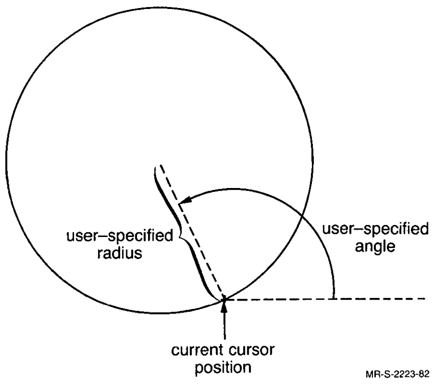
CIRCC

PURPOSE

CIRCC draws a circle using the current graphic cursor location as a point on the circumference. You supply a value for the radius and a value for the angle of the radius to the invisible horizontal line that passes through the current graphic cursor location.

After CIRCC draws the circle, the graphic cursor is left at its original location. See Figure 5–6.

Figure 5–6: Illustration of the CIRCC Subroutine



FORM

CIRCC (radius, angle)

ARGUMENT DESCRIPTION

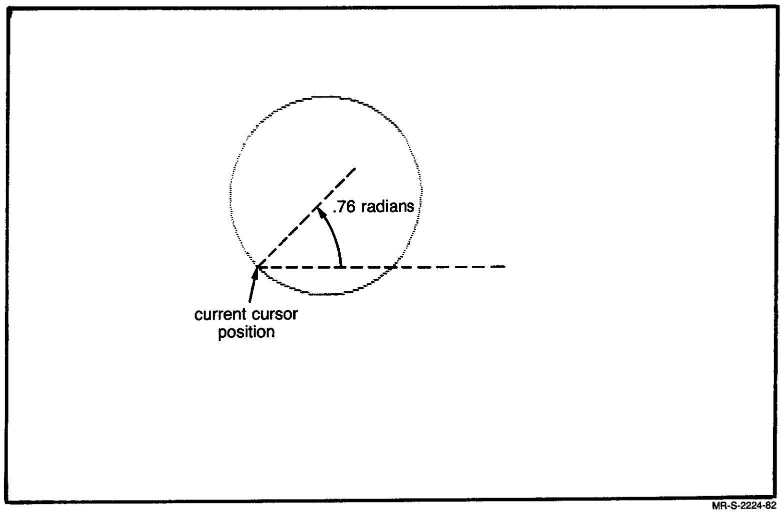
radius	is a floating point expression that defines the radius of the circle in world coordinates.
angle	is a floating point expression that defines the angle formed at the graphic cursor location between the radius and the horizontal. The unit of measure can be radians (default) or degrees. If the angle is positive, CIRCC draws in a counterclockwise direction; if the angle is negative, it draws clockwise.

EXAMPLE

```
CIRCC (100.0, .76)
```

draws a circle using the current graphic cursor location as the starting point for drawing the circumference of a circle. The circle has a radius of 100 units and the center of the circle is 0.76 radians (about 45 degrees) above the current graphic cursor location. The graphic cursor is left at its starting location. See Figure 5–7.

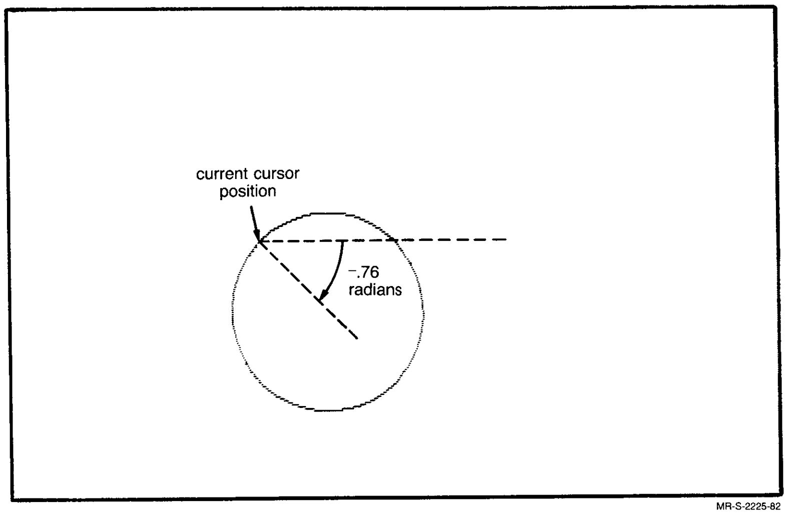
Figure 5–7: CIRCC's Placement of a 0.76 Radian Angle



```
CIRCC (125.0, -.76)
```

draws a circle using the current graphic cursor location as the starting point for drawing the circumference of a circle. The circle has a radius of 125 units. The center of the circle, at −0.76 radians, is below the graphic cursor. See Figure 5–8.

Figure 5–8: CIRCC's Placement of a −0.76 Radian Angle



RELATED SUBROUTINES

SDGREE and SRADNS determine whether the angle argument is interpreted as degrees or radians; SRADNS is the default.

SLNPAT determines the line pattern.

SSHADE creates shading for the arc.

CIRCLE and CIRCXY also draw circles, but the current graphic cursor location defines the center of the circle instead of a point on the circumference.

ARC and ARCC draw arcs.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

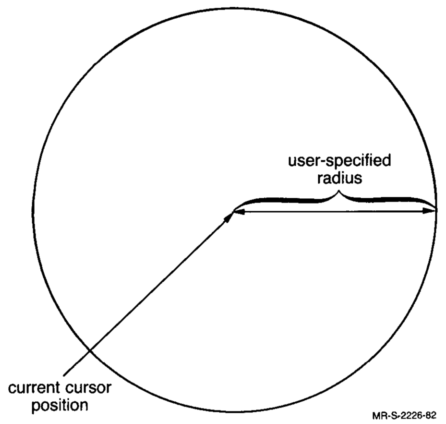
CIRCLE

PURPOSE

CIRCLE draws a circle using the current graphic cursor location as the center; you supply the value for a radius.

After the circle is drawn, the graphic cursor location is at its starting location in the center of the circle. See Figure 5–9.

Figure 5–9: Illustration of the Circle Subroutine



FORM

CIRCLE (radius)

ARGUMENT DESCRIPTION

radius	is a floating point expression that defines the radius of the circle in world coordinates.

EXAMPLE

```
CIRCLE (100.0)
```

draws a circle whose center is at the current graphic cursor location and whose radius is 100 units long. After the circle is drawn, the graphic cursor is again at the center of the circle.

RELATED SUBROUTINES

SLNPAT determines the line pattern.

SSHADE creates shading for the circle.

CIRCC also draws a circle, but the current graphic cursor location defines a point on the circumference of the circle instead of defining its center.

CIRCXY also draws a circle, but it requires that you specify a point on the circumference instead of specifying the radius.

ARC and ARCC draw arcs.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

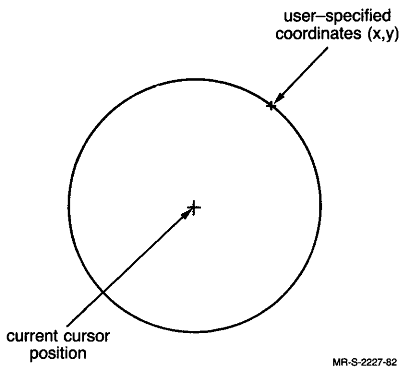
An example is given in Section 2.1, General Strategy for Drawing Graphic Objects.

PURPOSE

CIRCXY draws a circle using the current graphic cursor location as the center, and a point you specify as the starting location of the circumference.

After the circle is drawn, the graphic cursor is at its starting location at the center of the circle. See Figure 5–10.

Figure 5–10: Illustration of the CIRCXY Subroutine



FORM

CIRCXY (x, y)

ARGUMENT DESCRIPTION

x and y	are floating point expressions that define a location on the circumference of the circle in world coordinates.

EXAMPLE

```
CIRCXY (150.0, 200.0)
```

draws a circle whose center is at the current graphic cursor location and whose circumference passes through location (150., 200.). After the circle is drawn, the graphic cursor is left at the center of the circle.

RELATED SUBROUTINES

SLNPAT determines the line pattern.

SSHADE creates shading for the circle.

CIRCC draws a circle, but it uses the current graphic cursor location as a point on the circumference rather than as the center of the circle.

CIRCLE draws a circle, but it requires that you specify the radius rather than a point on the circumference.

ARC and ARCC draw circles.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

PURPOSE

CLRSCR (*Clear Scr*een) clears the graphic screen; that is, it clears all images that were created when the terminal was in graphic mode, including graphic text displayed by LINETX and TEXT. CLRSCR does not affect images created when the terminal was in text mode.

FORM

CLRSCR

ARGUMENT DESCRIPTION

none

EXAMPLE

```
CLRSCR
```

RELATED SUBROUTINES

CLRTXT erases the VT100 screen; that is, it erases images that were created when the terminal was in text mode.

RESTRICTIONS

CLRSCR erases all graphic images. To erase only selected ones, rewrite them in erase mode or complement mode.

ERROR MESSAGES

none

PROGRAM EXAMPLE

An example is given in Section 2.1, General Strategy for Drawing Graphic Objects, and in Section 2.1.1, Clearing the Screen.

CLRTXT

PURPOSE

CLRTXT (*Clear Text*) erases the VT100 screen; that is, it erases the images that were created with the terminal in text mode. It does not erase images created when the terminal is in graphic mode.

FORM

CLRTXT

ARGUMENT DESCRIPTION

none

EXAMPLE

```
CLRTXT
```

RELATED SUBROUTINES

CLRSCR erases the graphic screen.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

An example is given in Section 2.1, General Strategy for Drawing Graphic Objects, and in Section 2.1.1, Clearing the Screen.

PURPOSE

CPYSCR (*Copy Scr*een) transfers the entire contents of the graphic (VT125) screen to the LA34–VA printer, an optional device connected to the back of the VT125 by cable.

A call to CPYSCR can occur within a program to copy the screen for that program, or you can use it to create a 1–call, "stand–alone" program you can use any time to copy the screen.

FORM

CPYSCR

ARGUMENT DESCRIPTION

none

EXAMPLE

```
CPYSCR
```

RELATED SUBROUTINES

none

RESTRICTIONS

CPYSCR copies only the graphic images. It does not copy text written on the screen in regular text mode.

ERROR MESSAGES

none

PROGRAM EXAMPLE

The following program draws several graphic objects on the screen, then calls CPYSCR to copy them to the LA34–VA printer.

```
C  Initialize VT125
      CALL INITGR(5)
      CALL CLRSCR
      CALL CLRTXT
C
C  Draw various graphics objects on the screen
      CALL BOX(0.,0.,767.,479.)
      CALL BOX(200.,200.,400.,400.)
      CALL MOVE(384.,240.)
      CALL CIRCLE(200.)
      CALL POLYGN(3,150.)
      CALL LINETX (2,2,'This picture should appear on the LA34-VA.')
C
C  Now call CPYSCR to copy the VT125 graphics.
      CALL CPYSCR
C
      END
```

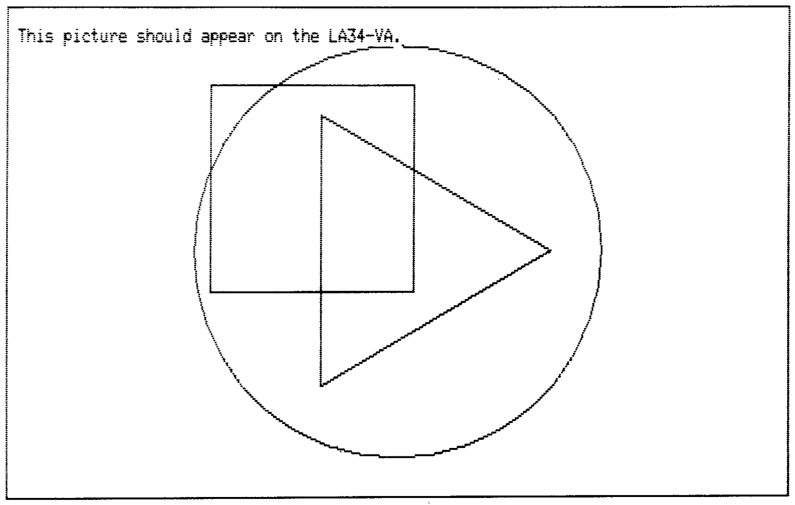


Figure 5–11: Pictures Illustrating the CPYSCR Program Example

DPAPER

PURPOSE

DPAPER (*D*raw *Paper*) clears the screen and creates a grid similar to a sheet of graph paper, on which to plot data. You can draw linear and/or logarithmic grids, can draw lines fully across the grid or indicate tickmarks along the four edges of the grid, and can select a color for the grid lines.

See Section 3.2, Creating the Graph Paper, for more information. For the parameters that DPAPER resets, see the NOTE at the beginning of Chapter 3.

FORM

DPAPER ({'xaxistype'}, {xa}, {xb}, {'yaxistype'}, {ya}, {yb}, {'gridcolor'})

ARGUMENT DESCRIPTION

'xaxistype'	is an optional string expression; it specifies the type of scale for the x–axis, and whether the axis will be gridded (lines fully drawn across the screen) or ungridded (tickmarks only). If unspecified, xaxistype defaults to LIN, a ungridded linear graph. Valid arguments are:

'LIN'	Ungridded linear
'LOG'	Ungridded logarithmic (base 10)
'GLIN'	Gridded linear
'GLOG'	Gridded logarithmic (base 10)

xa	is an optional integer expression; it specifies the number of cells (major divisions) along the x–axis. If 'xaxistype' is LOG or GLOG, the cells are cycles, xa specifies the number of cycles, and xa's default value is 3. Values 1 through 20 are valid. If 'xaxistype' is LIN or GLIN, xa's default value is 5. The recommended values for linear paper are multiples of 2 and 5, so that the rounding algorithm will give the desired results.
xb	is an optional integer expression; it signifies the number of minor, subcell, divisions within each major division along the x–axis. For example, if xb is 4, there are 3 lines drawn within the major division, creating four subdivisions. A value of 1 indicates that the subcell is coextensive with the cell. When the 'xaxistype' argument is LIN or GLIN and you want the number of subcells to be greater than one, the number of cells must be 20 or less; if the number of subcells can be one, you can have any number of cells. The number of subcells should always be 20 or less.

When the 'xaxistype' argument is LOG or GLOG, you can use only 1, 2, 5, 9, or 18.

The xb argument defaults to 5 for LIN and GLIN, and 9 for LOG and GLOG.

'yaxistype'	is an optional string expression; it specifies the type of scale for the y–axis, and whether the axis will be gridded or ungridded. Valid arguments are the same as for 'xaxistype'. The 'yaxistype' default is LIN.
ya	is an optional integer expression, specifying the number of cells along the y–axis. Values 1 through 20 are valid. If the 'yaxistype' is LOG or GLOG, ya defaults to 3. If 'yaxistype' is LIN or GLIN, it defaults to 5. A value of zero is invalid. The recommended values are multiples of 2 and 5, so that the rounding algorithm will give the desired results.
yb	is an optional integer expression; it signifies the number of subcells for each cell along the y–axis. When the 'yaxistype' is LIN or GLIN, the number of subcells should always be 20 or less. When 'yaxistype' is LOG or GLOG, it can take only 1, 2, 5, 9, or 18. yb defaults to 5 when LIN or GLIN is the 'yaxistype'; it defaults to 9 with LOG or GLOG.
'gridcolor'	is an optional string expression; it specifies the color of the grid lines, GRAY1, GRAY2, or GRAY3. Its default is the GRAY3.

EXAMPLE

```
DPAPER (,8,,,10,,)
```

draws an ungridded linear graph with eight cells on the x–axis, 10 cells on the y–axis, and five (default) subcells per cell on both axes. As you see in Figure 5–12, the cell boundaries show as long tick-marks, and the subcell boundaries show as smaller ones.

```
DPAPER ('GLIN',10,2,'LOG',2,9,'GRAY2')
```

draws a gridded linear x–axis with 10 cells that have two subcells each, and draws an ungridded logarithmic y–axis with two cells (cycles) that have nine subcells each. The grid lines are drawn in GRAY2. See Figure 5–13.

Figure 5–12: Ungridded Linear Graph Paper

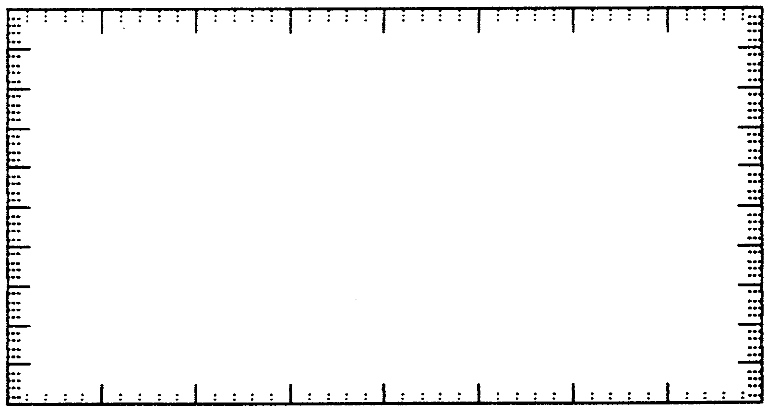
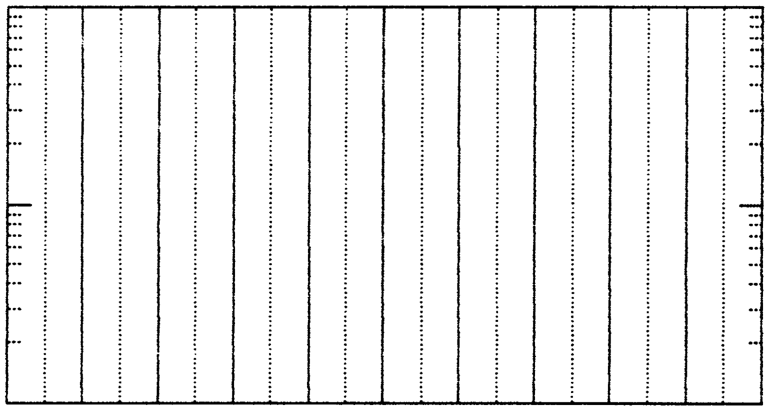


Figure 5–13: Gridded Linear X–Axis and Ungridded Logarithmic Y–Axis Graph Paper



RELATED SUBROUTINES

LNAXIS assigns numeric values to the cells of the specified axis, labels the cells with those values, and labels the axis itself with the text you supply.

LTAXIS assigns number values to the cells of the specified x–axis, displays a text label at each cell boundary, and labels the axis itself with the text you supply.

RESTRICTIONS

none

ERROR MESSAGES

After each of the following error messages is printed and you press the carriage return, the default value is supplied and the program continues.

1. Error Code 010

   ```
   %RGL-W-CNF, color 'x' not found
   ```

   means that you misspelled the gridcolor or referenced a nonexistent color.

2. Error Code 460

   ```
   %RGL-W-NCX, 'x' is an invalid number of cells (X axis)
   ```

   means that you specified a value for the xa argument that is not legal.

3. Error Code 470

   ```
   %RGL-W-NCY, 'x' is an invalid number of cells (Y axis)
   ```

   means that you specified a value for the ya argument that is not legal.

4. Error Code 500

   ```
   %RGL-W-NXT, 'x' is a nonexistent X axis type
   ```

   means that you specified a string expression for the xaxistype argument that does not match any value defined for that argument.

5. Error Code 510

   ```
   %RGL-W-NYT, 'x' is a nonexistent Y axis type
   ```

   means that you specified a string expression for the yaxistype argument that does not match any value defined for that argument.

6. Error Code 520

   ```
   %RGL-W-NSX, 'x' is an invalid number of subcells (X axis)
   ```

   means that you specified a value for the xb argument that is not legal for the scale to which the argument applies.

7. Error Code 530

   ```
   %RGL-W-NSY, 'x' is an invalid number of subcells (Y axis)
   ```

   means that you specified a value for the yb argument that is not legal for the scale to which the argument applies.

PROGRAM EXAMPLE

See DPAPER used in examples in Section 3.2, Creating the Graph Paper.

GCLOSE

PURPOSE

GCLOSE closes a file that was opened by a call to GSAVE.

FORM

GCLOSE

ARGUMENT DESCRIPTION

none

EXAMPLE

```
GCLOSE
```

RELATED SUBROUTINES

GSAVE opens a file whose contents will be all subsequent graphic commands. If you call GSAVE a second time, it closes that file and opens another one. Therefore, you can close a graphic file either by a call to GCLOSE or by a second call to GSAVE.

RESTRICTIONS

As a precautionary measure, your application program should close any open file before it exits.

ERROR MESSAGES

none

PROGRAM EXAMPLE

See the example in the GSAVE subroutine description.

GETLOC

PURPOSE

GETLOC (*Get Loc*ation) retrieves the coordinates of the current graphic cursor location and returns them in the x and y arguments.

You may want to use GETLOC:

- when you specified relative coordinates in your program and now you would like to know the absolute position of the cursor
- when you used a GLOAD command (which reads in a saved graphic file that may have changed the graphic cursor location) and you would now like to know where the RGL/11 subroutines last placed the graphic cursor
- when you wrote a message with graphic text, which leaves the graphic cursor at the end of the message, and you would like to know where that location is

FORM

GETLOC (x, y)

ARGUMENT DESCRIPTION

x and y	are floating point variables where GETLOC stores the coordinates of the current graphic cursor location; they are stored as world coordinates.

EXAMPLE

```
GETLOC (X, Y)
```

RELATED SUBROUTINES

LOCATE uses its own cursor, which the user can position, and returns the coordinates of that location.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

See the example in the MOVE subroutine description.

GETSTA

PURPOSE

GETSTA (*Get Sta*tus) returns in its argument the error code of the most recent error, if one has occurred. The initial value of the status variable is "001, the code for an error–free condition. It is set in all calls to INITGR and is set by GETSTA itself after the status variable has been read.

The initial error–handling procedure is that errors cause a message to be displayed on the screen and also cause an error code (in octal notation) to be stored in istat. If you call the SNDBUG (*S*et *N*o *Debug*) subroutine, the messages are not displayed, but the istat variable continues to be updated. By calling GETSTA at appropriate times, you can continue to get error information even when the terminal is in no–debug mode.

FORM

GETSTA (istat)

ARGUMENT DESCRIPTION

istat is an integer variable, which is a number in octal notation, representing the error status variable. The numbers and their corresponding messages are listed below (the identification "RGL–W–" that is displayed on the screen is omitted here):

Table 5–1: RGL/11 Error Codes

Error Code	Message
001	NORMAL, normal successful completion
010	CNF, color 'x' not found
100	INVWINCOORD, invalid window coordinates
110	OUTRANGE, coordinates out of range
150	INVCHARSET, 'x' is not a valid character set
160	INVSHADPAT, 'x' is not a valid shading pattern
170	INVTEXSIZ, 'x' is not a valid text size
200	INVLINE, 'x' is not a valid line pattern
210	INVMODE, 'x' is not a valid writing mode
220	INVSIDES, invalid number of sides
230	OPENIN, error opening 'x' as input
240	OPENOUT, error opening 'x' as output
250	RER, read error on unit 'x'
260	WER, write error on unit 'x'

continued on next page

Table 5–1: RGL/11 Error Codes (Cont.)

Error Code	Message
270	BADMARKER, 'x' is not a valid marker
330	NOCHARSET, character set 'x' has not been loaded
340	INVNUMEL, 'x' is an invalid number of elements
460	NCX, 'x' is an invalid number of cells (X axis)
470	NCY, 'x' is an invalid number of cells (Y axis)
500	NXT, 'x' is a nonexistent X axis type
510	NYT, 'x' is a nonexistent Y axis type
520	NSX, 'x' is an invalid number of subcells (X axis)
530	NSY, 'x' is an invalid number of subcells (Y axis)
560	IMM, 'x' is an invalid maximum and/or minimum
570	POW, all points are outside the window
600	AID, 'x' is an invalid axis identifier
610	TCC, 'x' is too many characters per cell
640	XNS, X axis has not been scaled
650	YNS, Y axis has not been scaled
740	NPL, logarithm of non–positive number
750	NOPAPER, no graph paper has been drawn
770	FIP, function is invalid with current graph paper

Appendix B, RGL/11 Error Messages, lists the subroutines that generate each error, and explains the circumstances of the error more completely.

EXAMPLE

```
GETSTA (ISTAT)
```

RELATED SUBROUTINES

SDEGUG enables error messages to be displayed on the screen.

SNDBUG stops display of error messages on the screen.

RESTRICTIONS

none

ERROR MESSAGES

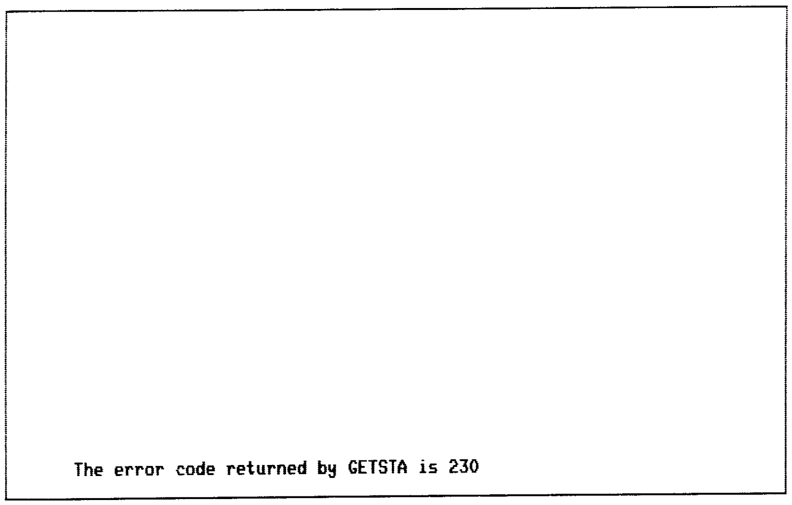
none

PROGRAM EXAMPLE

The following example produces an error by calling GLOAD with an invalid file name, then displays the error code returned by GETSTA (see Figure 5–14).

```
         INTEGER istat
C
C   Initialize VT125 and put a box around the screen.
         CALL INITGR(5)
         CALL CLRSCR
         CALL CLRTXT
         CALL BOX (0.,0.,767.,479.)
C
C   Call SNDBUG so that GETSTA is the only
C   source for error codes.
         CALL SNDBUG
C
C   Now call GLOAD with an invalid file name to produce an error.
         CALL GLOAD('INVALID.NAME',)
C
C   Call GETSTA to get the error code and
C   display it on the screen in octal form.
         CALL GETSTA(istat)
C
         TYPE 100,istat
100      FORMAT(10X,'The error code returned by GETSTA is ',O3)
C
         END
```

Figure 5–14: Picture Illustrating a GETSTA Error Code



```
The error code returned by GETSTA is 230
```

GLOAD

PURPOSE

GLOAD reads a graphic file saved by GSAVE and displays on the screen all the graphic images that are in the file. GLOAD reads the file in the device designated by the Logical Unit Number. The LUN's initial default value is 2.

After GLOAD reads in the graphic file, it resets the terminal's attributes to the values they had immediately prior to the GLOAD call, thus ensuring that the terminal's attributes will again be what the RGL/11 subroutines expect. (The file that GLOAD reads in contains ReGIS commands that affect the terminal, but do not affect the internal RGL/11 tables; therefore, the file could affect terminal attributes and throw off subsequent RGL/11 commands. GLOAD reestablishes the synchronization between the terminal's attributes and its internal RGL/11 state tables.)

FORM

GLOAD ('filespec', {LUN})

ARGUMENT DESCRIPTION

'filespec'	is a string expression that supplies the identification of the file you want loaded. See Preface, Documentation Conventions, for the arguments of a file specification.
LUN	is an optional argument; its value is an integer expression that specifies the Logical Unit Number of the device where the file is stored. The LUN you specify is used for the current and subsequent calls to GLOAD and LCHRST until you change it with another GLOAD or LCHRST call. The initial default LUN is 2. Do not use LUNs 1 or 5 (see Manual Conventions, in the Preface). If you do not specify a LUN, GLOAD uses the last LUN given by a call to GLOAD or LCHRST, and, if none, it then uses the initial default.

EXAMPLE

```
CALL GLOAD ('EUROPE.MAP', 4)
```

reads the contents (ReGIS graphic commands) of the EUROPE.MAP file from the device on Logical Unit Number 4.

RELATED SUBROUTINES

LCHRST, which loads character–set files, has a LUN argument. The LUN it names affects the default LUN of GLOAD, and vice versa.

GSAVE saves the files GLOAD reads.

RESTRICTIONS

If you do not supply a file extension, FORTRAN supplies a default extension of .DAT.

ERROR MESSAGES

1. Error Code `230`
   ```
   %RGL-W-OPENIN, error opening 'x' as input
   ```
   means that the file you specified could not be found. The call is ignored.

2. Error Code `250`
   ```
   %RGL-W-RER read error on unit 'x'
   ```
   means that the file could not be found on the device the LUN accessed, or it could not be read; for example, the file had no records in it or was an invalid type. The call is ignored.

PROGRAM EXAMPLE

See the Program Example in GSAVE.

GSAVE

PURPOSE

GSAVE opens a file and saves into it all ReGIS commands from the RGL/11 subroutines that you call subsequent to the GSAVE call. To end the GSAVE operation, either call GCLOSE to close the file or GSAVE which closes the open file and opens another one.

GSAVE does not affect the processing of the RGL/11 calls being saved; the calls still take effect normally, as well as being saved in the GSAVE file.

The initial default value for the Logical Unit Number is 3; the device addressed through the LUN is the device to which the file will be written.

FORM

GSAVE ('filespec', {LUN})

ARGUMENT DESCRIPTION

'filespec'	is a string expression that supplies the identification of the file GSAVE creates.
LUN	is an optional argument; it is an integer expression that specifies the Logical Unit Number of the device to which the file will be written. When you specify a LUN, that value is used in the current and subsequent GSAVE calls. Do not use LUNs 1 or 5 (see Manual Conventions in the Preface). If you do not specify a LUN, GSAVE uses the LUN specified in the last call to GSAVE, or, if there was no previous call to GSAVE, it uses the initial default of 3. If you specify a LUN that is out of range, GSAVE will default first to the previous GSAVE LUN, then to the initial default LUN, and will display a warning error message.

EXAMPLE

```
CALL GSAVE ('EUROPE.MAP',)
```

opens a file named EUROPE.MAP on logical unit 3. All subsequent ReGIS commands, generated by the RGL/11 subroutine calls you make, are saved in that file.

RELATED SUBROUTINES

GCLOSE closes a file opened by a call to GSAVE.

GLOAD retrieves a file saved by a call to GSAVE.

RESTRICTIONS

If you do not supply a file extension, FORTRAN supplies .DAT as the file extension.

Only one graphic file can be open at any one time. A second call to GSAVE closes the file that was open before opening a second file.

GSAVE does not save the RGL/11 subroutine calls; it saves only the ReGIS commands they generate. Some RGL/11 subroutines do not generate ReGIS strings and, therefore, they are not saved. Specifically, LFIXED, LFREE, LOCATE, SDEBUG, and SNDBUG are not saved.

ERROR MESSAGES

1. Error Code 240

   ```
   %RGL-W-OPENOUT, error opening 'x' as output
   ```

   means that you did not specify a valid file name or extension. The call is ignored; no file is created.

2. Error Code 260

   ```
   %RGL-W-WER, write error on unit LUN
   ```

   means that the RGL/11 software could not write a ReGIS string in the file because of a hardware error. This error is not caused by the call to GSAVE, but by RGL/11 subroutine calls that occur between a call to GSAVE and a call to GCLOSE. The RGL/11 subroutines might be able to draw the graphic object on the screen, but GSAVE will not be able to write the ReGIS string to the file.

PROGRAM EXAMPLE

The following program opens a file named EXAMPL.DAT, fills it with REGIS strings from RGL/11 subroutine calls, saves the file by a GCLOSE call, and uses one call to GLOAD to re–create the program on the screen.

```
C  Initialize VT125
         CALL INITGR(5)
         CALL CLRSCR
         CALL CLRTXT
         CALL SWINDO(0.,0.,1000.,625.)
C
C  All Regis commands between GSAVE and GCLOSE are saved.
         CALL GSAVE('EXAMPL.DAT',)
         CALL BOX(0.,0.,1000.,625.)
         CALL BOX(100.,110.,250.,200.)
         CALL MOVE(90.,225.)
         CALL TEXT('Write above the box')
         CALL GCLOSE
C
C  Now wait until the user types a <RETURN>.
         CALL LINETX(2,2,'Hit <RETURN> to continue.')
C
         ACCEPT 100
100      FORMAT(' ')
C
C  Clear the screen and then re-display the picture using GLOAD.
         CALL CLRSCR
         CALL GLOAD('EXAMPL.DAT',)
C
         END
```

Figure 5–15: Picture Illustrating a GSAVE Call, First Screen

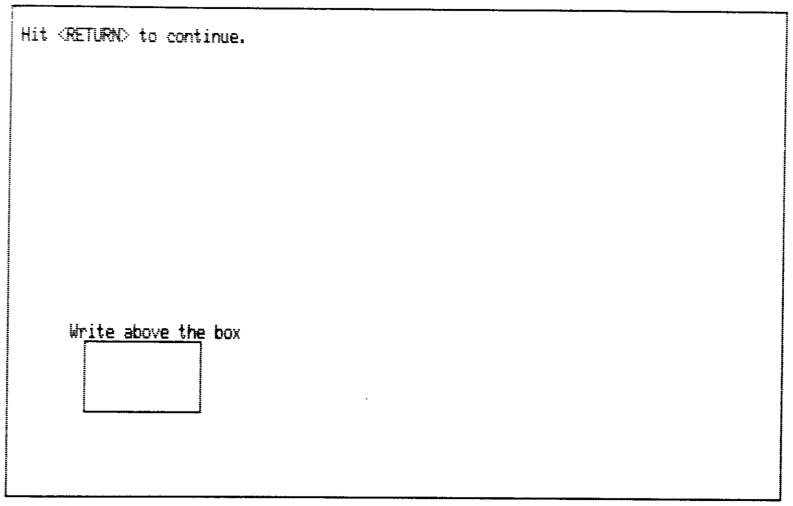
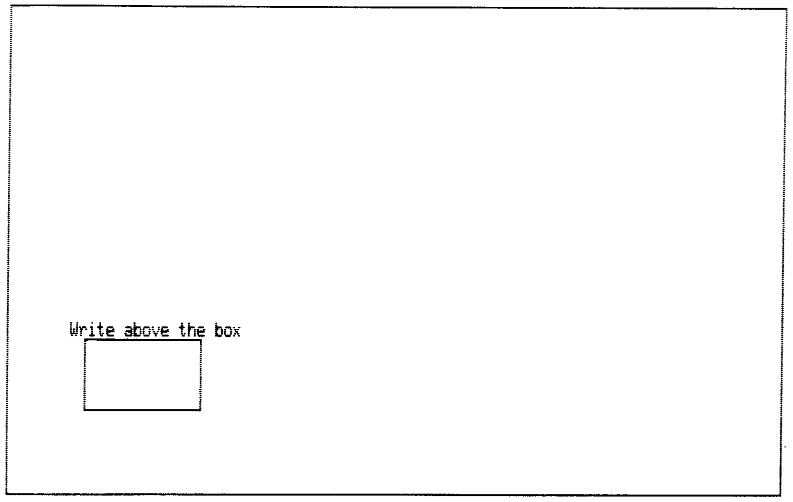

```
Hit <RETURN> to continue.
```

Figure 5–16: Picture Illustrating a GSAVE Call, Last Screen



PURPOSE

INITGR (*Init*ialize *Gr*aphics) sets all RGL/11 device characteristics to their initial default values. It must be the first graphic subroutine called by a graphic program.

The INITGR argument, LUN, lets you change the Logical Unit Number through which all graphic input and output is transmitted. The one you will probably use is 5, your terminal. All graphic input and output to and from the terminal is transmitted through this logical unit number: even if you change the LUN, you would continue to type your commands on your login terminal and error messages continue to be displayed there. Thus by changing the LUN, you can have one terminal to execute a program and another terminal to receive the output. See the Preface, Documentation Conventions, for LUN numbers.

The initial values INITGR supplies do the following:

- Set the physical screen coordinates to define the VT125 screen so that the (0, 0) location is at the lower left, the x axis has a range of 0 to 767, and the y–axis has a range of 0 to 479.
- Set the world coordinates equal to the screen coordinates.
- Set the graphic cursor at the (0,0) origin location, the bottom left of the graphic screen.
- Set the graphic text to the standard English character set and standard size.
- Set the line pattern to a solid line and the pattern multiplier to 2.
- Set writing mode to overlay and turns reverse image writing off (initialize mode).
- Set the shade pattern to the line pattern, sets the shade line to the bottom of the screen, and turns shading off.
- Set the angle unit of measure to radians.
- Enable the displaying of error messages on the screen.
- Set the value in the error–status variable (ISTAT, accessible by the GETSTA subroutine) to "001".
- Assign GRAY0 (dark) to color number 0, GRAY1 (dark gray) to 1, GRAY2 (light gray) to 2, and GRAY3 (white) to 3. On an optional color monitor, GRAY0 is dark, GRAY1 is blue, GRAY2 is red, and GRAY3 is green.
- Set color number 3 as the drawing color.
- Set hardware smoothing off.

FORM

INITGR (LUN)

ARGUMENT DESCRIPTION

LUN	is an integer expression that specifies the Logical Unit Number of the graphic input and output device. Ordinarily use 5, your terminal. If you use another number so you can use another device, put OPEN and CLOSE statements in your program; INITGR does not affect the device itself.

EXAMPLE

```
INITGR (5)
```

resets all device–independent characteristics to their predefined values and establishes the device addressed by LUN 5 as the input and output device for RGL/11 graphic commands. You continue to use your login terminal to input commands and to receive error messages.

RELATED SUBROUTINES

GSAVE's first subroutine call should be to INITGR so that the saved program begins with the terminal's attributes set to known values. Otherwise, it begins with the attributes that are current at the time GLOAD reads the file in, and results are unpredictable.

RESTRICTIONS

INITGR does not clear the screen. If you want to clear the screen, you must call CLRSCR to clear graphic objects, and CLRTXT to clear text–mode images.

INITGR does not open or close the device the LUN specifies; you must do that within your program (see the PROGRAM EXAMPLE below).

ERROR MESSAGES

1. Error Code `260`
   ```
   %RGL-W-WER, write error on unit 'x'
   ```

PROGRAM EXAMPLE

The following program fragment shows you how to specify a LUN other than 5 in your program.

```
OPEN (UNIT=10, NAME='TT41:')
CALL INITGR (10)
  .
  .
  .
CLOSE (UNIT=10)
```

Another example is in Section 2.1, General Strategy for Drawing Graphic Objects.

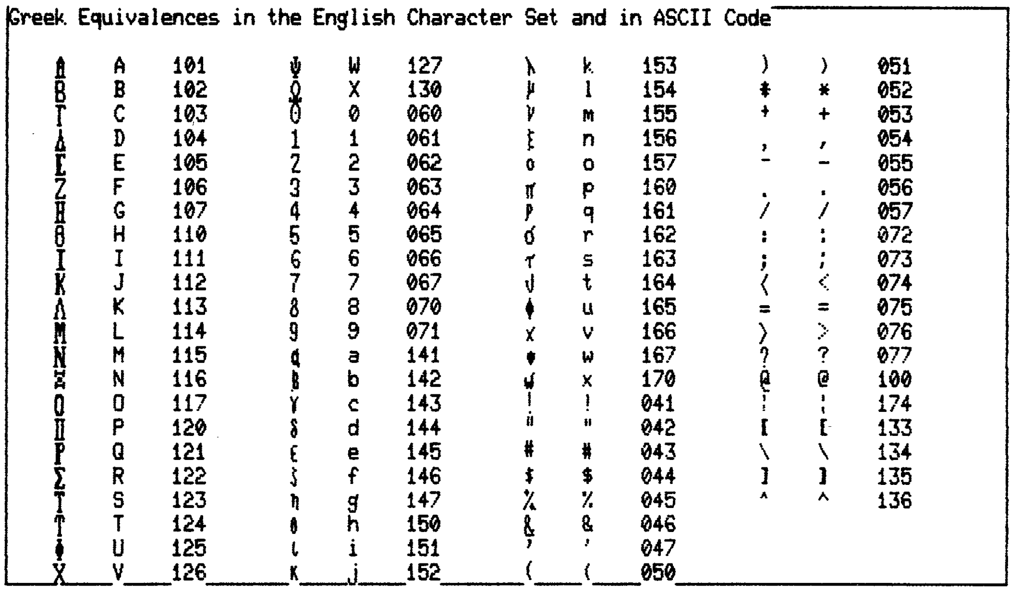
PURPOSE

LCHRST (*Load Character Set*) loads the alternate character set — Greek characters — into the terminal's memory from a specified file. The character set is identified by the number 1. After you call LCHRST, you must call SCHRST to activate the character set.

The default character set is the standard ASCII set. To return to it after using the Greek set, you need to use only SCHRST; the default set is permanently loaded in the terminal's memory.

To access the characters in the Greek character set, you type either their English equivalents or their ASCII code. Table 5–2 lists these values.

Table 5–2: Greek Equivalences in the English Character Set and in ASCII Code



Greek Equivalences in the English Character Set and in ASCII Code

Α	A	101	Ψ	W	127	λ	k	153	)	)	051
Β	B	102	Ω	X	130	μ	l	154	*	*	052
Γ	C	103	0	0	060	ν	m	155	+	+	053
Δ	D	104	1	1	061	ξ	n	156	,	,	054
Ε	E	105	2	2	062	ο	o	157	-	-	055
Ζ	F	106	3	3	063	π	p	160	.	.	056
Η	G	107	4	4	064	ρ	q	161	/	/	057
Θ	H	110	5	5	065	σ	r	162	:	:	072
Ι	I	111	6	6	066	τ	s	163	;	;	073
Κ	J	112	7	7	067	υ	t	164	<	<	074
Λ	K	113	8	8	070	φ	u	165	=	=	075
Μ	L	114	9	9	071	χ	v	166	>	>	076
Ν	M	115	α	a	141	ω	w	167	?	?	077
Ξ	N	116	β	b	142	ψ	x	170	@	@	100
Ο	O	117	γ	c	143	!	!	041	|	|	174
Π	P	120	δ	d	144	"	"	042	[	[	133
Ρ	Q	121	ε	e	145	#	#	043	\	\	134
Σ	R	122	ζ	f	146	$	$	044	]	]	135
Τ	S	123	η	g	147	%	%	045	^	^	136
Υ	T	124	θ	h	150	&	&	046			
Φ	U	125	ι	i	151	'	'	047			
Χ	V	126	κ	j	152	(	(	050			

FORM

LCHRST (number, 'filespec', {LUN})

ARGUMENT DESCRIPTION

number	is an integer expression that references the character set to be loaded. The only valid number is 1, for the Greek character set that is on your distribution volume.
'filespec'	is a string expression that identifies the file where the character set is stored.

LUN is an optional argument; it is an integer expression that specifies the Logical Unit Number of the device you want LCHRST to read the file from. The LUN you specify will also be used as the default for subsequent calls to LCHRST and GLOAD.

If you do not specify a LUN, the LUN last specified in a GLOAD or LCHRST call is used, or, if no LUN was specified, LCHRST uses LUN 2 as the initial default. If you specify a LUN that is out of range, LCHRST uses the previous GLOAD or LCHRST LUN, or, if none, the initial LUN default, and it displays a warning error message on the screen.

EXAMPLE

```
LCHRST (1, 'GREEK.FNT', 3)
```

loads the file GREEK.FNT, the Greek character set, into character set 1 from LUN 3.

RELATED SUBROUTINES

SCHRST activates the character set you load with LCHRST.

The currently active character set is displayed on the screen when you call any of the following subroutines: LINETX, LNAXIS, LTAXIS, SSHADE, TEXT, and PDATA.

GLOAD has a LUN argument; the LUN it names can affect the default LUN of LCHRST and vice versa.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 150
   ```
   %RGL-W-INVCHARSET, 'x' is not a valid character set
   ```
   means that the character set number was not a 1. The call is ignored; the previous font continues to be used.

2. Error Code 230
   ```
   %RGL-W-OPENIN, error opening 'x' as input
   ```
   means that the file you specified either could not be found or could not be read. The call is ignored; the previous font continues to be used.

3. Error Code 250
   ```
   %RGL-W-RER, read error on unit LUN
   ```
   means that the file could not be read. The call is ignored; the previous font continues to be used

PROGRAM EXAMPLE

The following program fragment loads the Greek character set into alternate character set 1 in the terminal's memory, and writes five characters of the set. The current default value for the LUN is used.

```
CALL LCHRST (1, 'GREEK.FNT', )
CALL SCHRST (1)
CALL TEXT ('abcde')
```

LFIXED

PURPOSE

LFIXED (*L*ocate *Fixed*) enables you or the user of your program to guide a special locator cursor along the data path of a graph and find the coordinates of up to ten points by typing a numeric key (1, 2,...0). After you or the user of your program selects the points, LFIXED returns an array of integers, called the index array, that points into the X and Y data arrays that define the plot.

When your program calls LFIXED, the locator cursor is displayed on the terminal screen over the first point on the data path. (The locator cursor consists of two straight lines configured as a crosshair with a blinking, diamond–shaped polygon in the center.) To move the locator cursor, you press the left– or right–arrow key. The left–arrow key moves the cursor backward along the data path from one point to the next, and the right–arrow key moves it forward. Do not hold an arrow key down in order to move the cursor continuously; instead press the key repeatedly. To move the cursor faster, press the key labeled PF4 on the auxiliary keypad. Thereafter the cursor moves over 10 data points each time you press an arrow key. To return it to the slower rate, press the PF3 key.

To get a pointer value into the index array, you press one of the numeric keys on the main keyboard, 1 through 0. (Type only the number; the RETURN key or any other key is not necessary unless you want to terminate the LFIXED sequence.) After you have located the cursor where you want it, press 1. That puts an integer into the index array at element 1 that indicates the point in the array; that is, this integer points to the locations in the xarray and yarray that contain the coordinates for that location. Then move the locator cursor to the second location you want, press 2 (which loads the second element of the index array), and so on. Type 0 to load the tenth element.

To terminate LFIXED, type any key except the left and right arrow keys, or the numeric keys on the main keyboard. After you type a terminating key, control returns to the calling program.

FORM

LFIXED (number, {xarray}, yarray, {'yaxis'}, indexarray)

ARGUMENT DESCRIPTION

number	is an integer expression that specifies the number of points in the data path you want to select from. It must be less than or equal to the smaller of the two arrays (xarray or yarray). If it is not, results are indeterminate.
xarray	is an optional argument; it is the name of the floating–point array that contains the x–coordinates of the data path along which the locator cursor will

move. It must be the same array name as in the PDATA subroutine that is displaying the plot. Its default values are sequential integers from one to n.

yarray	is the name of the floating–point array that contains the y–coordinates of the data path. It must be the same array name as in the PDATA subroutine that created the graph.
'yaxis'	is an optional single–character string expression that indicates the y–axis scale to which the yarray argument applies. Valid forms for this argument are: 'L' (for left y–axis) or 'R' (for right y–axis). Its default value is conditional on whether the y–axis (axes) is scaled:

- If only the right y–axis is scaled, the default is 'R,' and program continues.
- If only the left y–axis is scaled, the default is 'L,' and program continues.
- If both left and right are scaled, the default is 'L,' and program continues.
- If neither one is scaled, the default is 'L,' error 650, axis not scaled, is invoked, and control returns to the calling program.

indexarray	is the name of an integer array to be filled with numbers that act as pointers into the xarray and yarray that created the graph. You must dimension the indexarray to contain ten elements. LFIXED zeroes the elements in the array before using it. Therefore any element that is still zero when LFIXED terminates was never loaded.

EXAMPLE

```
LFIXED (30, XDATA, YDATA,,INDEX)
```

sets up the INDEX array so that when you move the locator cursor and strike a numeric key, LFIXED will put into INDEX a number that points into the XDATA and YDATA arrays to get the coordinates for the location of the locator cursor. The 30 indicates that the first 30 elements of the arrays specified can be accessed by the locator cursor. The default y–axis is the y–axis that was scaled.

RELATED SUBROUTINES

LFREE also displays the locator cursor. It returns x and y coordinates of any point within the graph, not necessarily on the data path. The coordinates are based on the scale of a graph's axes.

LOCATE uses the locator cursor for picture–drawing subroutines. It returns the world coordinates of a single location.

RESTRICTIONS

If the data arrays are not scaled to the graph or vice versa so that they produce locations outside the graph's scale, LFIXED does not work properly; it produces error 640 (x–axis problem) or 650 (y–axis problem). If the xarray is in random order, the first point of the data path may not be the left–most point.

ERROR MESSAGES

1. Error Code 340

   ```
   %RGL-W-INVNUMEL, 'x' is an invalid number of elements
   ```

   means that you specified the number of elements in an array to be less than or equal to zero. RGL/11 immediately returns control to the calling program.

2. Error Code 570

   ```
   %RGL-W-POW, all points are outside the window
   ```

   means that all the data points passed to LFIXED lie outside the data–plotting window.

   When this error occurs, LFIXED terminates and returns control to the calling program.

3. Error Code 600

   ```
   %RGL-W-AID, 'x' is an invalid axis identifier
   ```

   means that you specified an axis identifier that does not match either of the codes defined for 'yaxis' ('L' or 'R').

   When this error occurs, the yaxis assumes the default value and the program continues.

4. Error Code 640

   ```
   %RGL-W-XNS, x-axis has not been scaled
   ```

   means that you did not scale the x–axis by a call to LNAXIS, LTAXIS or PDATA. When this error occurs, LFIXED terminates and returns control to the calling program.

5. Error Code 650

   ```
   %RGL-W-'YNS, y-axis has not been scaled
   ```

   means that you did not scale the y–axis by a call to either LNAXIS, LTAXIS, or PDATA. When this error occurs, LFIXED terminates and returns control to the calling program.

6. Error Code 750

   ```
   %RGL-W-NOPAPER, no graph paper has been drawn
   ```

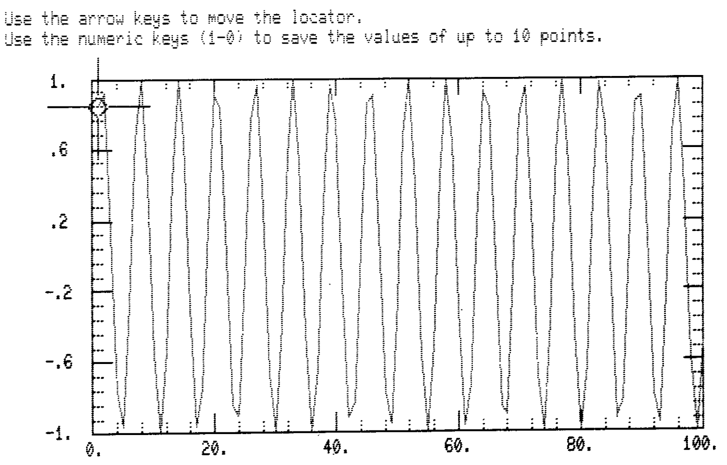
means that you did not call DPAPER or PDATA before calling LFIXED. LFIXED terminates and returns control to the calling program.

PROGRAM EXAMPLE

The following program draws a sine wave, then calls LFIXED. The user can enter up to 10 points into the index array using LFIXED. The points selected are then marked using PPOINT.

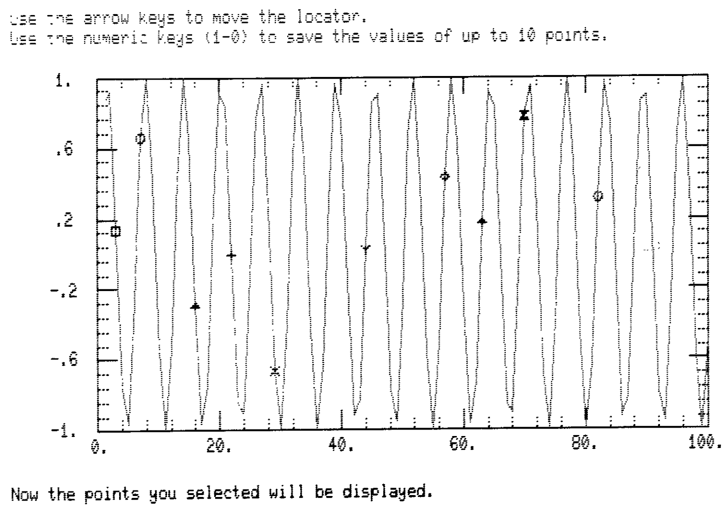
```
        REAL xarray(100), yarray(100)
        INTEGER index(10)
C
C  Initialize VT125
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
C
C  Put a sine wave into the x and y arrays and plot the results.
        DO 100 i = 1,100
          xarray(i) = FLOAT (i)
          yarray(i) = SIN (xarray(i))
100     CONTINUE
        CALL PDATA (100,xarray,yarray,' ',' ',,,,,)
C
C  Now use LFIXED to pick off up to ten points on the graph.
        CALL LINETX(1,1,'Use the arrow keys to move the locator,')
        CALL LINETX(2,1,'Use the numeric keys (1-0) to save the values
        1 of up to 10 points.')
        CALL LFIXED (100,xarray,yarray,'L',index)
C
C  Now put a marker at the coordinates of the user selected points
C  using PPOINT.
        CALL SCOLOR (' ',3)
        CALL LINETX(24,1,'Now the points you selected will be
        i displayed,')
        DO 300 i = 1,10
          n = index(i)
          IF (n .EQ. 0) GOTO 200
            x = xarray(n)
            y = yarray(n)
            CALL PPOINT(x,y,'L',i)
200       CONTINUE
300     CONTINUE
C
        END
```

Figure 5–17: **Picture Illustrating First Screen of LFIXED Program**



```
Use the arrow keys to move the locator.
Use the numeric keys (1-0) to save the values of up to 10 points.
```

Figure 5–18: **Picture Illustrating Final Screen of LFIXED Program**



```
Use the arrow keys to move the locator.
Use the numeric keys (1-0) to save the values of up to 10 points.
```

```
Now the points you selected will be displayed.
```

LFREE

PURPOSE

LFREE "reads" the coordinates of a point in a graph. You select the coordinates by means of a locator cursor, and LFREE puts them into x and y variables. The coordinates are returned in terms of the scale for the axes.

When your program calls LFREE, the locator cursor is displayed at the center of the plotting area. You move the locator cursor by pressing the arrow keys, up, down, left, or right. Each time you press one of these keys, the locator cursor moves by 10 pixels. If you want to slow it down to a 1–pixel move, press the *PF3* key on the auxilliary keypad. To speed it up again to 10–pixel moves, press the *PF4* key. Do not hold an arrow key down in order to move the cursor continuously; instead press the key repeatedly.

When you have moved the locator cursor to the location whose coordinates you want, press any key on the keyboard except the arrow keys, SHIFT, ESC, and DELETE keys. The x and y variables receive the new values, LFREE terminates, and the locator cursor is erased from the screen. The DELETE key returns the coordinates of the center position of the data plotting window; SHIFT has no effect; do not use ESC.

FORM

LFREE (x, y, key, {'yaxis'})

ARGUMENT DESCRIPTION

x and y	are names of floating–point variables that receive the x– and y–coordinates of the locator cursor's location at the time you terminate LFREE. They are defined in terms of the x– and y–axis scales, respectively, that were defined in the previous LNAXIS, LTAXIS, or PDATA call.
key	is the name of a byte variable. When you press a key to terminate LFREE, LFREE puts that key's ASCII code in the key argument.
'yaxis'	is an optional single–character string expression to specify the y–axis whose coordinates you want. Valid arguments are: 'L' (left y–axis) or 'R' (right y–axis). Its default value is conditional on which y–axis (axes) is scaled:

- If only the right y–axis is scaled, the default is 'R,' and program continues.
- If only the left y–axis is scaled, the default is 'L,' and program continues.
- If both left and right are scaled, the default is 'L,' and program continues.

- If neither one is scaled, the default is 'L,' error 650, axis not scaled, is invoked, and control returns to the calling program.

EXAMPLE

```
LFREE (XCOORD, YCOORD, KEY, 'R')
```

puts the x–coordinate of the current locator cursor location in XCOORD, and the y–coordinate in YCOORD; the y–coordinate is determined in relation to the right y–axis scale. The ASCII value of the key that terminates LFREE is put into the byte variable KEY.

RELATED SUBROUTINES

LFIXED also displays the locator cursor; it returns up to 10 coordinate pairs of locations on a data path.

LOCATE displays the same locator cursor, but for the picture–drawing subroutines. It returns world coordinates of a single location.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 600

   ```
   %RGL-W-AID, y-axis is an invalid axis identifier
   ```

   means that you specified an axis identifier that does not match either of the codes defined for 'yaxis' ('L' or 'R').

   When this error occurs, the yaxis defaults to the y–axis that is scaled and the program continues. (If no y–axis is scaled, it generates error 650 described below.)

2. Error Code 640

   ```
   %RGL-W-XNS, 'xaxis' has not been scaled
   ```

   means that you did not call either the LNAXIS, LTAXIS, or PDATA subroutine to scale the x–axis. When this error occurs, LFREE terminates and returns control to the calling program.

3. Error Code 650

   ```
   %RGL-W-'YNS, 'yaxis' has not been scaled
   ```

   means that you did not call either the LTAXIS, LNAXIS, or PDATA subroutine to scale the y–axis LFREE referenced. When this error occurs, LFREE terminates and returns control to the calling program.

4. Error Code 750

   ```
   %RGL-W-NOPAPER, no graph paper has been drawn
   ```

means that you did not call DPAPER or PDATA before calling LFREE. LFREE terminates and returns control to the calling program.

PROGRAM EXAMPLE

The following program calls LFREE ten times and prints each coordinate pair and the value of Key at the terminal. It also displays each coordinate pair on the graph using PPOINT.

```
      REAL x,y
      LOGICAL*1 Key
C
C  Initialize VT125
      CALL INITGR(5)
      CALL CLRSCR
      CALL CLRTXT
C
C  Draw graph paper and scale the axes
      CALL DPAPER(,,,,,,)
      CALL LNAXIS('XT','X DATA',0.,100.,)
      CALL LNAXIS('YL','Y DATA',0.,100.,)
C
C  Now call LFREE 10 times and display the user selected coordinates
C  and a marker at those coordinates using PPOINT.
      DO 200 i=1,10
        CALL LFREE(x,y,Key,'L')
        CALL CLRTXT
        TYPE 100,i,x,y,Key
100     FORMAT(' ','Coords, #',i2,' = ',2(F6.2,','),' Key = ',A1)
        CALL PPOINT(x,y,'L',i)
200   CONTINUE
      END
```

Figure 5–19: Picture Illustrating LFREE, First Screen

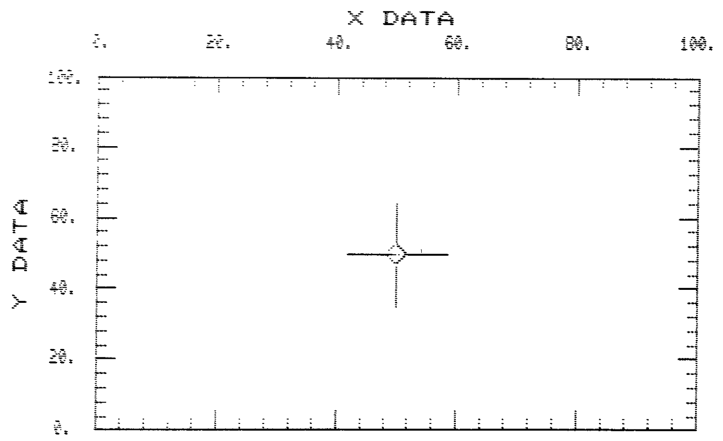
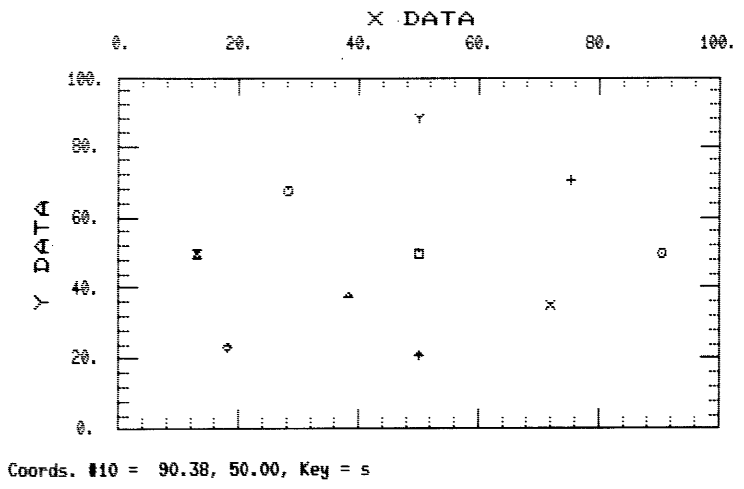


Figure 5–20: Picture Illustrating LFREE, Last Screen



```
Coords. #10 =  90.38, 50.00, Key = s
```

PURPOSE

LINE draws a line from the current graphic cursor location to the specified location.

After the line is drawn, the graphic cursor is at the end of the line.

FORM

LINE (x, y)

ARGUMENT DESCRIPTION

x and y	are floating point expressions that define the end-point of the line in world coordinates.

EXAMPLE

```
LINE (100.0, 100.0)
```

draws a line from the current graphic cursor location to location (100.,100.). The graphic cursor is left at location (100.,100.).

RELATED SUBROUTINES

SLNPAT determines the line pattern.

RELLIN also draws a line, but its arguments are the distances in the x and y directions rather than the destination's coordinates.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 100
   ```
   %RGL-W-INVWINCOORD, invalid window coordinates
   ```
   means that you tried to move the cursor more than 32767 pixels. The call is ignored.

PROGRAM EXAMPLE

The following program fragment sets the window, moves the cursor to location (100.,125.), then draws a line from there to (300.,350.). The graphic cursor is left at location (300.,350.).

```
CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
CALL MOVE (100.0, 125.0)
CALL LINE (300.0, 350.0)
```

Another example of LINE, with an illustration, is in the RELMKR subroutine description.

LINETX

PURPOSE

LINETX (*Line Text*) displays in graphic text the message you specify beginning at the row and column you specify. There are 24 rows and 80 columns. Numbering begins at the screen's origin location, the upper left corner of the screen. The message is always displayed in the standard size and in replace mode. You can use either reverse mode, for reverse writing, or no–reverse mode; the default is no–reverse. LINETX can display text strings of alternate character sets, such as GREEK.FNT.

After LINETX writes the message, it returns the graphic cursor to the location it had before the call.

FORM

LINETX (irow, icol, 'string')

ARGUMENT DESCRIPTION

irow and icol	are integer expressions that define the row and column where the message begins. The screen has 24 rows, numbered 1–24 from top to bottom, and 80 columns, numbered 1–80 from left to right.
'string'	is a string expression containing the message to be displayed. The length of the string can be 80 characters or less; any message longer than 80 characters is truncated at the 80th character. If you want a double quotation mark displayed within the text string, you must use two double quotation marks (" "); this is true whether you pass the message as a string literal or as a string variable. If you want an apostrophe (as in "call's"), you must use two in a row ("). The string is written in replace mode.

EXAMPLE

```
LINETX (10, 20, 'Say " "HELLO" " ')
```

displays the message:

Say "HELLO"

beginning at row 10, column 20 of the screen.

RELATED SUBROUTINES

SCHRST determines the character set (standard or Greek) that LINETX will use.

CLRSCR erases LINETX messages.

TEXT also displays a message, but the message begins at the current graphic cursor location, and the size of the characters can be changed to the size specified in a call to STXSIZ. (STXSIZ does not affect LINETX.)

RESTRICTIONS

If a message extends beyond the window, the message is automatically clipped on the window boundary and no error message is generated.

Any message over 80 characters is truncated at the 80th character. If you want to use a series of quotation marks (for example, as a border), you can use only 40 on a line since you must use two double quotation marks to get one.

The characters of the message are always standard size; STXSIZ does not affect LINETX displays.

ERROR MESSAGES

1. Error Code 110

   ```
   %RGL-W-OUTRANGE, coordinates out of range
   ```

   means that irow is either less than 1 or more than 24, or that icol is either less than 1 or more than 80. The call is ignored; no message is displayed.

PROGRAM EXAMPLE

See LINETX used in the example in Section 2.7, Retrieving Location Coordinates.

LNAXIS

PURPOSE

LNAXIS (*L*abel *N*umeric *Axis*) enables you to scale and label an axis.

It supplies numeric values for the cells (major divisions established by the DPAPER subroutine) of the axis you specify, and labels the cell boundaries with scaled numbers. You can make the values either exact or rounded numbers, and you can control whether the scale will be user–scaled or autoscaled. When LNAXIS prints the numeric labels, it uses floating–point notation. If a label exceeds six characters, LNAXIS prints the label in exponential notation with the exponent printed on the following line. LNAXIS also enables you to label the axis as a whole.

An application program should call LNAXIS once for each axis that requires a scale. For example, it would call LNAXIS twice to scale two y–axes and once to scale an x–axis.

Section 3.3, Scaling and Labeling the Axes, describes ways to use the subroutine and gives sample programs.

FORM

LNAXIS ('axisid', {'axislabel'}, {minvalue}, {maxvalue}, {exact})

ARGUMENT DESCRIPTION

'axisid'	is a two–character string expression that specifies the axis you want to scale and label. Valid strings are: 'XT' — upper x–axis 'XB' — bottom x–axis 'YL' — left y–axis 'YR' — right y–axis
'axislabel'	is an optional string expression that gives a title to the whole axis; it is printed centered and adjacent to the specified axis. This label contains information about the axis function, the scaling variable, or other information about the axis. There can be 40 or less characters in an x–axis label, and 20 or less in a y–axis label. Its default value is a blank label.
minvalue	is an optional floating–point expression that specifies the value of the first cell on the axis. This argument controls autoscaling. If it is unspecified or if you make minvalue equal to maxvalue, LNAXIS performs autoscaling when you next call PDATA. Otherwise, it scales according to the minvalue and maxvalue range. The minvalue argument also provides the label that is displayed at the first cell

	boundary, if exact numbers are requested. The first cell boundary is at the left side of the first cell. There is always one more label than there is cell. For example, when there are three cells, there are four cell labels.
maxvalue	is an optional floating–point expression that specifies the value of the last cell on the axis. If it is unspecified or if you set maxvalue equal to minvalue, LNAXIS performs autoscaling when you next call PDATA. Otherwise, it scales the axis according to the range in minvalue and maxvalue. The maxvalue argument also provides the label that is displayed at the last cell, if exact numbers are requested.
exact	is an optional logical expression that determines whether LNAXIS generates rounded numbers for labels or not. When the argument is ".TRUE.," LNAXIS generates numeric labels based on exactly what is implied by the minvalue and maxvalue arguments. When the argument is ".FALSE.," LNAXIS generates rounded numbers for the label. The default value is false, for rounded number labels. The exact argument can be true only for linear, not logarithmic, axes. For log axes, the exact argument is ignored; you always get rounded numbers.

EXAMPLE

```
LNAXIS ('XB','X-AXIS VALUES',0.0,10.0,.TRUE.)
```

labels the bottom x–axis with numbers between 0 and 10. The scale is user–scaled, and the cell labels are exact. If the x–axis has 10 cells, the labels will be integers from 0 to 10; otherwise, LNAXIS generates numeric labels in equal increments based on the number of cells along the axis. It also gives the bottom x–axis the character label "X–AXIS VALUES." The ".TRUE." argument implies a linear scale, since it cannot be used with logarithmic scales.

```
LNAXIS ('YR','Y-AXIS VALUES',3.55,12.469,)
```

labels the right y–axis with a set of rounded numbers (the default) derived from the given data, and uses a user–specified scale of 3.55 to 12.469 for the axis. It also gives the axis the label "Y–AXIS VALUES."

RELATED SUBROUTINES

DPAPER creates the graph paper to be labeled.

LTAXIS scales and labels the cell divisions along a graph's axis, and displays alphanumeric labels at each cell boundary (not just numeric labels, as LNAXIS does). It also gives the entire axis a label.

RESTRICTIONS

If the minvalue or maxvalue argument is equal to or less than .01 or equal to or greater than 10,000, LNAXIS prints the label in exponential notation with the exponent printed on the following line.

ERROR MESSAGES

1. Error Code 560

   ```
   %RGL-W-IMM, 'x' is an invalid maximum and/or minimum
   ```

   means that you specified an illegal number for either the minvalue argument, the maxvalue, or both. When this error occurs, LNAXIS attempts to write the label and allows PDATA to autoscale after the error message is displayed.

2. Error Code 600

   ```
   %RGL-W-AID, 'x' is an invalid axis identifier
   ```

   means that you specified a code for 'axisid' that was not 'XT', 'XB', 'YL', or 'YR'. LNAXIS terminates and returns control to the calling program.

3. Error Code 740

   ```
   %RGL-W-NPL, logarithm of non-positive number
   ```

   means that you specified either a negative number or a 0.0 for the minvalue or maxvalue argument when you were labeling a logarithmic axis. When this error occurs, LNAXIS immediately returns control to the calling program.

4. Error Code 750

   ```
   %RGL-W-NOPAPER, no graph paper has been drawn
   ```

   means that you did not call DPAPER before calling LNAXIS. If this error occurs, control returns immediately to the calling program.

5. Error Code 770

   ```
   %RGL-W-FIP, function is invalid with current graph paper
   ```

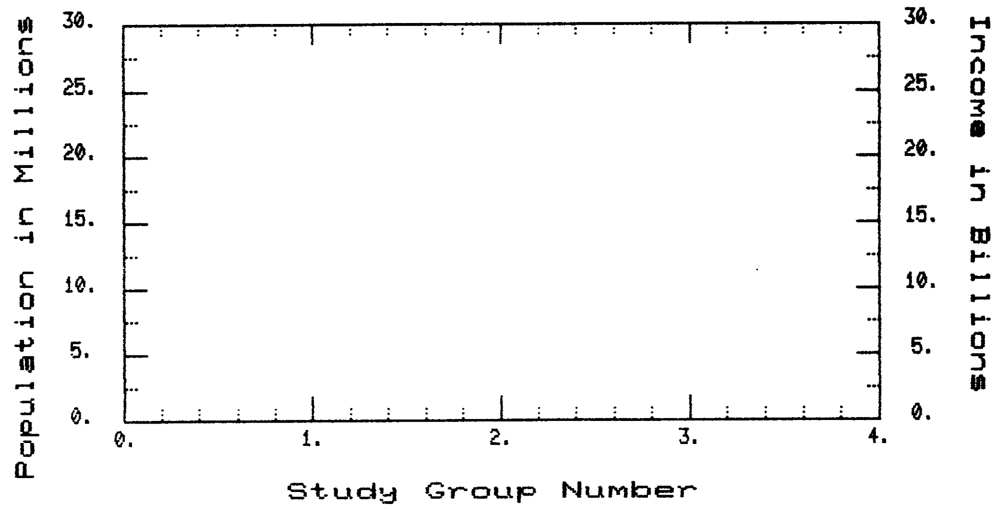
   means that the axis you specified is already scaled. When this error occurs, LNAXIS terminates and returns control to the calling program.

PROGRAM EXAMPLE

The following program fragment calls DPAPER to draw in GRAY2 a linear grid with four cells and five subcells along the x–axis, and six cells and two subcells along the y–axis. It then calls LNAXIS three times to scale and label the left and right y–axes and the bottom x–axis. The graph uses user–specified scaling and exact cell labels. Figure 5–21 shows the graph that is displayed.

```
CALL DPAPER ('LIN',4,5,'LIN',6,2,'GRAY2')
CALL LNAXIS ('YL','Population in Millions',0.,30.,,TRUE.)
CALL LNAXIS ('YR','Income in Billions',0.,30.,,TRUE.)
CALL LNAXIS ('XB','Study Group Number',0.,4.,,TRUE.)
```

Figure 5–21: Graph that Illustrates Labeling and Scaling by LNAXIS



LOCATE

PURPOSE

LOCATE displays a special cursor, a "*locator cursor*", on the screen. It handles instructions from the keyboard to move the locator cursor around on the screen, and stores the coordinates of its location in response to a keyboard inquiry. The locator cursor is two lines, crossed, with a blinking diamond–shaped polygon in the center; it is always displayed in GRAY3.

When your program calls LOCATE, the locator cursor is displayed on the screen at the x and y coordinate you specify, or, if you do not specify a location, at the current graphic cursor location. If location of the graphic cursor is off the screen, the locator cursor is brought up on the closest edge of the screen.

You move the locator cursor by pressing the arrow keys, up, down, left, or right. Each stroke moves the locator cursor by 10 pixels; If you want to move the cursor slower, press the PF3 key on the auxiliary keypad; thereafter the cursor moves by one pixel each time you press an arrow key. To return to the faster cursor movement, press the PF4 key. Do not hold an arrow key down in order to move the cursor continuously; instead press the key repeatedly. If you try to move the cursor beyond the edge of the screen, it stays at the edge until you move it back toward the center.

When the locator cursor is where you want it and you want to know the coordinates of that location, you type any key on the keyboard except the DELETE key or the arrow keys. LOCATE puts the coordinates in the x and y variables, erases the locator cursor, and terminates its operation. To terminate the LOCATE operation without getting the location, type the DELETE key.

Refer to Section 2.7, Retrieving Location Coordinates, for more information.

FORM

LOCATE (x, y, key)

ARGUMENT DESCRIPTION

x and y — are floating point expressions that can be used in two ways: (1) they can define the starting location for the locator cursor in world coordinates, or (2) they can store the coordinates of the current location of the locator cursor. If you want to define the starting position of the locator cursor, you must define x and y before calling LOCATE.

When you type any key but a DELETE or an arrow key, LOCATE replaces the contents of x and y with the world coordinates of the locator cursor location and then terminates its operation. If you type the

	DELETE key, the contents of x and y are not changed, and the operation terminates.
key	is a byte variable into which LOCATE puts the ASCII value of the terminating character.

EXAMPLE

```
LOCATE (X, Y, Key)
```

displays the locator cursor at the current graphic cursor location. You can move the cursor to the location you want and press any key so that LOCATE puts the coordinates of that location into the x and y variables, and terminates.

RELATED SUBROUTINES

GETLOC returns the current location of the graphic cursor.

LFIXED and LFREE work with the data–plotting subroutines in the same way as LOCATE works with the picture–drawing subroutines.

RESTRICTIONS

If you or the user of your application program aborts the LOCATE sequence by returning to the monitor (rather than to the calling program) while the terminal is displaying the locator cursor, the terminal is left in graphic mode and does not respond to the monitor. To recover, you must execute the "reset" SET–UP function (described in the *VT125 User Guide*). You may want to disable user commands that return to the monitor while the LOCATE sequence is in operation.

ERROR MESSAGES

none

PROGRAM EXAMPLE

The following program calls LOCATE and checks for a terminating character of either DELETE or RETURN. It displays the x and y coordinates selected, and the key used.

```
        REAL x,y
        LOGICAL*1 Key
C  Initialize VT125 and draw some graphics objects.
        CALL INITGR (5)
        CALL CLRSCR
        CALL CLRTXT
        CALL BOX (0.,0.,767.,479.)
        CALL MOVE (384.,240.)
        CALL CIRCLE (200.)
        CALL BOX (200.,200.,400.,400.)
C
        x = 350.
        y = 215.
10      CONTINUE
20      CONTINUE
        CALL LOCATE (x,y,Key)
```

```
C
C  The cursor is displayed at (350., 215.); the execution
C  of the program does not continue until the user types a
C  key. If the user presses a DELETE key, (177 octal), the
C  program loops so the user can continue selecting locations
C  without displaying values on the screen.
C
       IF (key .EQ. "177) GOTO 20
C
C  Otherwise, the program types the location coordinates
C  and the KEY's contents.
C
       TYPE 100,x,y,key
100    FORMAT(' The location is ',2F8.2,' and the KEY is ',A1)
C
C  If the key that is typed is not the RETURN key (15 octal),
C  the program gets another location. If it is RETURN, the
C  program leaves the loop and exits.
C
       IF (key .NE. "015) GOTO 10
C
       END
```

Other LOCATE examples are in the OBJECT.FOR program in Section 2.1, General Strategy for Drawing Graphic Objects, and in Section 2.7, Retrieving Location Coordinates.

LTAXIS

PURPOSE

LTAXIS (*L*abel *T*ext *Axis*) assigns a set of numeric values to the cell (major) divisions of a graph's x– or y–axis, labels those divisions with alphanumeric labels of equal length, and gives the whole axis a character label, which can serve as the axis title or the graph title.

Section 3.3.4, Labeling the Cells of an Axis, describes ways to use this subroutine. A demonstration program, named LABEL2.FOR is on your distribution medium.

FORM

LTAXIS ('axisid', {'axislabel'}, minvalue, maxvalue, {maxchars}, {'string'})

ARGUMENT DESCRIPTION

'axisid'	is a string expression that specifies the axis you want to label. Valid strings are: 'XT' – upper x–axis 'XB' – bottom x–axis 'YL' – left y–axis 'YR' – right y–axis
'axislabel'	is an optional string expression that serves as the title of the axis or of the graph. LTAXIS centers this title and places it adjacent to the axis specified by the 'axisid' argument. The string for the x–axis can contain 40 characters or less. LTAXIS writes in overlay mode. The default value is a blank label. The string for the y–axis can be 20 characters or less. If a title exceeds these limits, the middle of the title is centered and the left and right ends of the title are clipped.
minvalue	is a floating–point expression that specifies the value (not the label) of the first cell on the axis. If this argument is unspecified or if you make minvalue equal to maxvalue, autoscaling is invoked and the scales are not displayed until PDATA is called. Otherwise, if you specify the minvalue and maxvalue, the scales are in exact values and are immediately displayed.
maxvalue	is a floating–point expression that specifies the value (not the label) of the outer boundary of the last cell on the axis. If this argument is unspecified or if you make minvalue equal to maxvalue, autoscaling is invoked and the scales are not displayed until PDATA is called. Otherwise, if you specify the minvalue and maxvalue, the scales are in exact values and are immediately displayed.

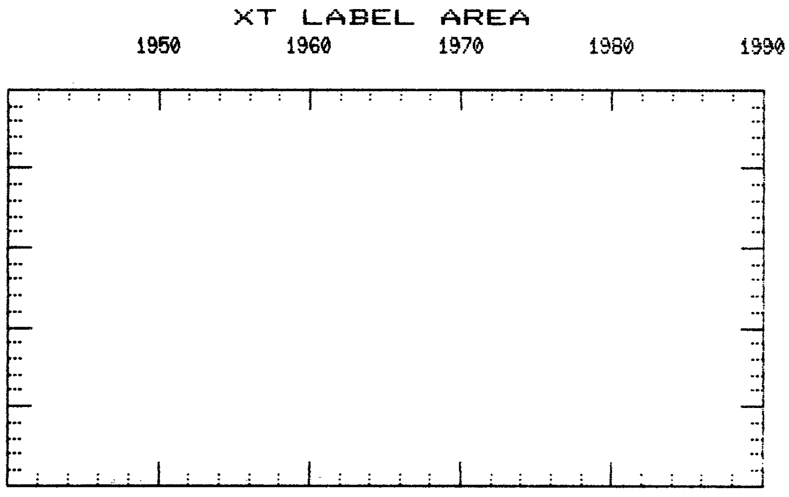
maxchars	is an optional integer expression; it specifies the number of characters from the 'string' argument to be printed at each cell boundary. Its maximum value varies with the number of labels, but for the y–axis only, the maximum number of characters printed at each cell boundary is 6. The number of labels is always one more than the number of cells. The number of labels times "maxchars" cannot exceed 64. The default value is 0, no characters printed.
'string'	is an optional string expression that consists of all the cell–boundary labels. The labels are concatenated in the 'string' argument; LTAXIS displays each label centered about each cell boundary in groups of characters maxchars long. There is one more label than there are cells; the first label is displayed at the left boundary of the first cell. This argument is defaultable if you default the maxchars argument or give it a value of 0.

EXAMPLE

```
LTAXIS ('XT','XT LABEL AREA',0.,5.,4,'    19501960197019801990')
```

labels the upper x–axis with the title XT LABEL AREA, scales the cells of the x–axis from 0.0 to 5.0, and establishes 4 as the number of characters in the cell–boundary labels. The first four characters of the cell–label string are blank, so that the first visible label is displayed under the first cell's second boundary. The labels 1950, 1960, 1970, 1980, and 1990 are centered under each succeeding cell boundary. See Figure 5–22.

Figure 5–22: Graph Illustrating the LTAXIS Example



RELATED SUBROUTINES

DPAPER creates the grid whose axes are to be labeled.

LNAXIS assigns a set of numeric values to the cell divisions along an axis, displays those values at each cell boundary, and gives the entire axis a character label that identifies its purpose.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 560

   ```
   %RGL-W-IMM, 'x' is an invalid maximum and/or minimum
   ```

   means that you specified a minvalue argument that was greater than or equal to the maxvalue argument. When this occurs, LTAXIS terminates and returns control to the calling program. If this is the only call to scale the axis, PDATA will autoscale when it is called.

2. Error Code 600

   ```
   %RGL-W-AID, 'x' is an invalid axis identifier
   ```

   means that you specified a code for 'axisid' that was not YL, YR, XT, or XB. LTAXIS terminates and returns control to the calling program.

3. Error Code 610

   ```
   %RGL-W-TCC, 'x' is too many characters per cell
   ```

   means that the number of labels along the axis times the value for "maxchars" exceeds 64. When this error occurs, the maxchars argument defaults to the largest number of characters that are possible to write at each cell boundary without any of the labels overwriting each other. The first 64 letters of the string argument are used.

4. Error Code 740

   ```
   %RGL-W-NPL, logarithm of non-positive number
   ```

   means that you specified either a negative number or a zero for the minvalue or maxvalue argument when you were labeling a logarithmic axis. When this error occurs, LTAXIS immediately returns control to the calling program.

5. Error Code 750

   ```
   %RGL-W-NOPAPER, no graph paper has been drawn
   ```

   means that you did not call DPAPER before calling LTAXIS. Control is returned to the calling program.

6. Error Code 770

   ```
   %RGL-W-FIP, function is invalid with current graph paper
   ```

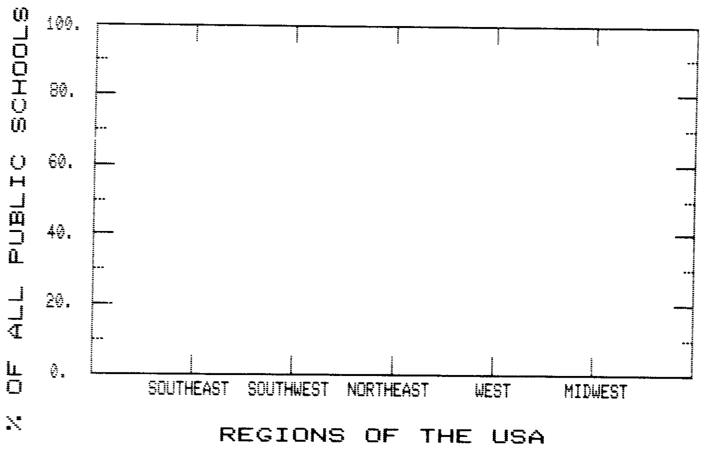
   means that the axis you specified is already scaled. When this error occurs, LTAXIS terminates and returns control to the calling program.

PROGRAM EXAMPLE

The following program fragment calls DPAPER to draw a linear graph in GRAY2 with six cells and one subcell along the x–axis and five cells and two subcells along the y–axis. It then calls LNAXIS to scale and label the left y–axis, labeling it "% OF ALL PUBLIC SCHOOLS," scaling it from 0.0 to 100.0, and using exact values for the cell–boundary labels. Next it calls LTAXIS to scale the bottom x–axis, titling it "REGIONS OF THE USA," and to display labels of nine characters in length at each of the cell boundaries along the bottom x–axis. Notice there are six labels. The first one is nine blank spaces; the others are "SOUTHEAST", "SOUTHWEST", "NORTHEAST", "WEST", and "MIDWEST".

```
      CALL DPAPER ('LIN',6,1,'LIN',5,2,'GRAY2')
      CALL LNAXIS ('YL','% OF ALL PUBLIC SCHOOLS',0.,100.,,.TRUE.)
      CALL LTAXIS ('XB','REGIONS OF THE USA',0.,6.,9,
     1 '         SOUTHEASTSOUTHWESTNORTHEAST   WEST    MIDWEST ')
```

Figure 5–23: Second Graph Illustrating LTAXIS Call



MARKER

PURPOSE

MARKER displays a marker at the location you specify in such a way that the marker's center is on that location. The default marker is a dot, marker number 0.

After the marker is drawn, the graphic cursor is left at that location.

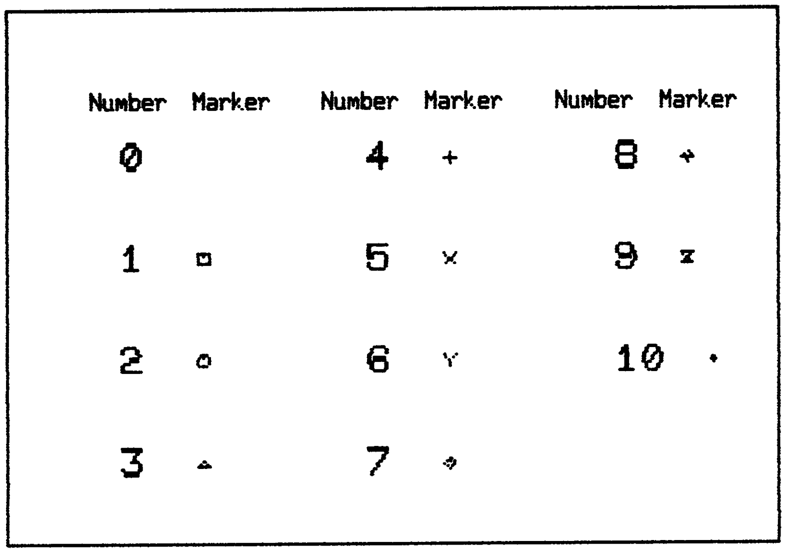
FORM

MARKER ({number}, {x}, {y})

ARGUMENT DESCRIPTION

number — is an optional argument; it is an integer expression from 0 through 10 that identifies the marker you want to display. If you do not specify a number, MARKER displays marker number 0, a dot, by default. The markers and their identifying numbers are:

Table 5–3: Markers and Their Identifying Numbers



Number	Marker	Number	Marker	Number	Marker
0		4	+	8	✢
1	□	5	×	9	⧗
2	○	6	Y	10	•
3	▲	7	◆		

x and y — are optional floating point expressions that define in world coordinates the location where you want the marker displayed. If you do not specify a value for x, MARKER uses the x–coordinate of the current graphic cursor location; if you do not specify a value for y, it uses the y–coordinate of the current graphic cursor location.

EXAMPLE

```
MARKER (2, ,)
```

displays an octagon that is centered on the current graphic cursor location; that location remains unchanged.

```
MARKER (3, 200.0, 300.0)
```

displays a triangle whose center is at location (200., 300.). The location of the graphic cursor is moved to (200., 300.).

RELATED SUBROUTINES

RELMKR also displays a marker, but its arguments are the distances in the x– and y–directions from the current graphic cursor location, rather than the coordinates of the desired location.

PPOINT displays a marker on a graph.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 270

   ```
   %RGL-W-BADMARKER, n is not a valid marker
   ```

   means that you specified a marker number that is less than 0 or greater than 10. RGL/11 uses the default marker and continues.

PROGRAM EXAMPLE

See program example in Section 2.6, Marking Locations.

MOVE

PURPOSE

MOVE moves the graphic cursor to the specified location. It does not draw a graphic object of any kind.

FORM

MOVE (x, y)

ARGUMENT DESCRIPTION

x and y	are floating point expressions that define the new location of the graphic cursor in world coordinates.

EXAMPLE

```
MOVE (100.0, 100.0)
```

moves the graphic cursor to location (100.0, 100.0).

RELATED SUBROUTINES

RELMOV also moves the graphic cursor, but its arguments are distances in the x– and y–directions from the current graphic cursor location, rather than the coordinates for the destination location.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code `100`
   ```
   %RGL-W-INVWINCOORD, invalid window coordinates
   ```
   means that you tried to move the cursor more than 32767 pixels. The call is ignored.

PROGRAM EXAMPLE

The following program fragment sets the window, moves the graphic cursor to location (100., 125.), and writes a message beginning at that location. The graphic cursor is left at the end of the message, and the GETLOC call puts the coordinates of that location into the X and Y variables and displays them on the screen.

```
      CALL SWINDO (0.0,0.0,1000.0,625.0)
      CALL MOVE (100.0,125.0)
      CALL TEXT ('Write starting at location (100.,125.)')
      CALL GETLOC (x,y)
      TYPE 100,x,y
100   FORMAT (' The current cursor location is ',2(F6.2,1X))
```

PDATA

PURPOSE

A single call to PDATA (*P*lot *D*ata) can create linear graphs and plot data from stored files. You can also use PDATA in combination with other data–plotting subroutines. By using it with just DPAPER, you can draw and plot logarithmic graphs; using it with DPAPER, LNAXIS, and LTAXIS, you can exercise full control over the grid, the numeric and text labels you supply for each axis, and other data plotting options.

If a DPAPER call does not precede a PDATA call, PDATA clears the screen and draws the grid. PDATA accepts 10 input arguments. The first three arguments specify the data arrays you want to plot and their size. The other seven arguments specify data plotting options such as the y–axis to be used, the color and type of the data line, the marker of the data points, and whether or not the smoothing and shading functions are in effect.

Section 3.1 describes using only PDATA to create graphs. Sections 3.2 to 3.8 describe using it in combination with other subroutines. For the parameters that PDATA resets, see the NOTE at the beginning of Chapter 3.

FORM

PDATA (n, {xarray}, yarray, {'yaxis'}, {'colorname'}, {marker}, {linetype}, {smooth}, {shade}, {yvalue})

ARGUMENT DESCRIPTION

n	is an integer expression that specifies the number of points you want to plot. The x– and y–arrays must contain at least this number of elements. If they do not, results are indeterminate.
xarray	is the name of an optional floating–point array that contains the values for the x–axis coordinates. If xarray is unspecified and the x–axis is autoscaled, xarray values default to sequential integers from one to n. (The x–axis is autoscaled when PDATA is used as the only data–plotting subroutine, or with certain LNAXIS calls.) If xarray is unspecified and the x–axis is user–scaled, PDATA supplies values for the xarray by subtracting "minvalue" from "maxvalue" and dividing the answer by the value in the n argument. This formula ensures that the points to be plotted are spread evenly across the grid.
yarray	is the name of a floating–point array that contains the values for the y–axis coordinates. When the graph paper you are using has two y–axes, you specify which y–axis is to be used with these values by the next argument, 'yaxis.'

'yaxis'	is an optional single–character string expression that specifies which y–axis scale PDATA is to use. Valid arguments are 'L' (for left y–axis) or 'R' (for right). Its default value is conditional on which y–axis (axes) is scaled:

- If only the right y–axis is scaled, the default is 'R,' and program continues.
- If only the left y–axis is scaled, the default is 'L,' and program continues.
- If both left and right are scaled, the default is 'L,' and program continues.
- If neither one is scaled, the default is 'L,' autoscaling is subsequently invoked, and the program continues.

'colorname'	is an optional string expression that specifies the color of the data line. Valid arguments are 'GRAY0', 'GRAY1', 'GRAY2', and 'GRAY3'. The default is GRAY3. (See Section 3.7.)
marker	is an optional integer expression that specifies what marker to display at each of the points to be plotted. The MARKER subroutine lists the eleven possible markers. The marker argument can assume a value in either of two ranges:

0 to 10	(displays the marker at every point)
100 to 110	(displays the marker at every 10th point)

If this argument is defaulted, no marker is displayed. If you plan to plot a large number of points, it is strongly recommended that you default the marker argument. By doing so, you speed up execution of the graph.

linetype	is an optional integer expression that specifies the line pattern connecting the points being plotted. It can be any number from 0 through 9, representing the line patterns explained in the SLNPAT subroutine. If you do not want the points connected, specify a linetype of 0.

The linetype default is 1, a solid line, when shading is disabled. When shading is enabled, its default is a special line pattern. Each time you call PDATA with linetype unspecified and shading enabled, the line pattern changes. After four such calls to PDATA, the special line patterns repeat. (This feature is recommended for shaded graphs; it gives increased legibility to the graph's data.)

smooth	is an optional logical expression that determines whether the points being plotted are connected by a smooth curve or whether they are connected by a series of straight lines. When smooth is ".TRUE.", the plot line forms a smooth curve; when it is false, the plot line is a series of connected straight lines. Its default is ".FALSE.", straight lines.
shade	is an optional logical expression that determines whether the area under the plot line is shaded. When shade is ".TRUE.", PDATA shades the area between the curve and the user–specified shade line on the y–axis (this shade line is determined by the next argument). The shade pattern PDATA uses is the one specified in the linetype argument. When the shade argument is ".FALSE.", PDATA does not shade the area under the curve. Its default value is ".FALSE.", shading disabled.
yvalue	is an optional floating–point argument that specifies the shade line to which the shade pattern is drawn. Its default value is the minimum value on the appropriate y–axis scale (the y–axis specified in the 'yaxis' argument).

EXAMPLE

```
PDATA (10,,YVALS,'L','GRAY2',1,,,,)
```

plots 10 points whose x–axis coordinates are the integers from 1 to 10 (default), and whose y–axis coordinates are values in the YVALS array. The y–axis coordinates are plotted with respect to the left y–axis. The points are marked with marker type 1 (a square), and connected by a solid line (default); the line color is GRAY2. Shading and smoothing functions are disabled.

```
PDATA (200,XVAL(100),YVAL(100),'R',,105,6,.TRUE.,,.TRUE.,)
```

plots 200 points defined by array elements 100 to 300 from each input array. The y–axis coordinates are plotted against the right y–axis. Marker 5 (a cross) marks every tenth point. The points are connected by a smooth curve in line pattern 6, and the shading function is enabled.

RELATED SUBROUTINES

PPOINT plots a single data point to an existing graph.

RESTRICTIONS

You cannot use more than two colors to plot data. The screen color and the grid color use two of the four color numbers, leaving two color numbers to receive the values of the colorname argument. If you attempt to use more than two colors, each call after the second one changes the color of the points plotted two calls before it.

When smoothing is enabled, the x–axis coordinate values must be passed from the input array in ascending order. If they are not, PDATA can produce a curve that follows an erratic path. Another restriction that smoothing imposes is that it prevents clipping from occurring at the top x–axis. It is possible that a line between two points may pass outside of the data plotting window and not be clipped.

If the shading operation is enabled when a plot is clipped at the top x–axis, (or when smoothing causes the arc to appear above the x–axis), PDATA does not shade the region between the two points that intersect the top x–axis.

ERROR MESSAGES

1. Error Code 010
   ```
   %RGL-W-CNF, color 'x' not found
   ```
   means that you misspelled the color name or referenced a nonexistent color name. The color defaults to the current drawing color and the program continues.

2. Error Code 200
   ```
   %RGL-W-INVLINE, 'x' is not a valid line pattern
   ```
   means that you specified a value for the linetype argument that was not from 0 through 9. When this error occurs, the argument assumes the default value and the program continues.

3. Error Code 270
   ```
   %RGL-W-BADMARKER, 'x' is not a valid marker
   ```
   means that you specified a value for the marker argument that was not from 0 through 10 or from 100 through 110. When this error occurs, the argument assumes the default value and the program continues.

4. Error Code 340
   ```
   %RGL-W-INVNUMEL, 'x' is an invalid number of elements
   ```
   means that you specified a value for the number argument that was zero or negative. Control returns to the calling program.

5. Error Code 600
   ```
   %RGL-W-AID, 'x' is an invalid axis identifier
   ```

means that you specified a value for the yaxis argument that was not "L" or "R." When this error occurs, one of three actions occur, depending on the condition of the graph:

- If both y–axes are scaled or just the left y–axis is scaled, PDATA uses "L" as the default.
- If only the right y–axis is scaled, PDATA uses "R" as the default.
- If neither y–axis is scaled, PDATA uses "L" as the default and autoscales the left y–axis.

6. Error Code 640

```
%RGL-W-XNS, X axis has not been scaled
```

means that one of two conditions occurred:

- Autoscaling was invoked but all values in the xarray are the same.
- The x–axis is logarithmic, autoscaling is invoked, but the xarray contains at least one non–positive number.

When this error occurs, PDATA terminates and returns control to the calling program.

7. Error Code 650

```
%RGL-W-YNS, Y axis has not been scaled
```

means that one of two conditions occurred:

- Autoscaling is invoked but all values in the yarray are the same.
- The y–axis is logarithmic, autoscaling is invoked, but the yarray contains at least one non–positive number.

When this error occurs, PDATA terminates and returns control to the calling program.

8. Error Code 740

```
%RGL-W-NPL, logarithm of non-positive number
```

means that an input array contained a non–positive coordinate for a logarithmic axis. When this error occurs and autoscaling is not in effect, it does not interrupt the data plotting sequence. However, the point will not be plotted. When this error occurs and autoscaling is in effect, control is immediately returned to the calling program because autoscaling requires all data points to be valid.

PROGRAM EXAMPLE

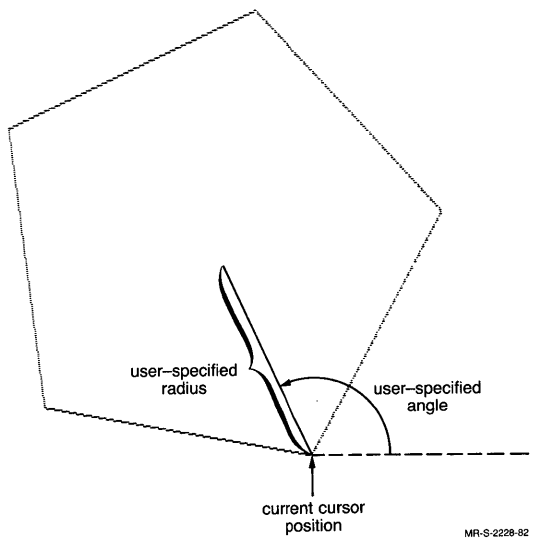
See Chapter 3 for several examples.

PURPOSE

POLYGC draws a regular polygon according to your specifications for the number of sides, the radius, and the angle that the polygon will rotate around one vertex. The current graphic cursor location defines the location of the vertex. A non–rotated polygon is symmetrical about a horizontal line which passes through the current graphic cursor location.

After the polygon is drawn, the graphic cursor is again at its starting location, the vertex. See Figure 5–24.

Figure 5–24: Illustration of the POLYGC Subroutine



FORM

POLYGC (nsides, radius, angle)

ARGUMENT DESCRIPTION

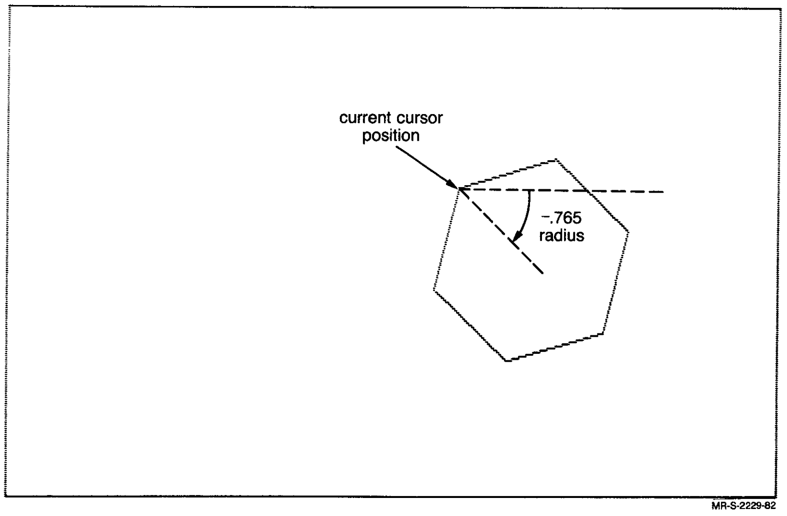
nsides	is an integer expression that defines the number of sides of the polygon. This argument must be greater than two.
radius	is a floating point expression that defines the distance in world coordinates from the center to any vertex of the polygon.
angle	is a floating point expression that defines the angle about which the polygon is to rotate. You can specify the angle in the positive (counterclockwise) direction or in the negative (clockwise) direction. The angle's unit of measure can be radians (default) or degrees.

EXAMPLE

```
POLYGC (6, 100.0, -.765)
```

draws a hexagon with its vertices 100 units from the center of the hexagon. The current graphic cursor location defines the location of the first vertex to be drawn; the -0.765 radian angle indicates that the vertex is located above and to the left of the hexagon's center. (The angle is formed in a clockwise direction from the horizontal line drawn through the current location of the graphic cursor because the angle is negative.) After the hexagon is drawn, the graphic cursor is again at its starting location, the first vertex. See Figure 5–25.

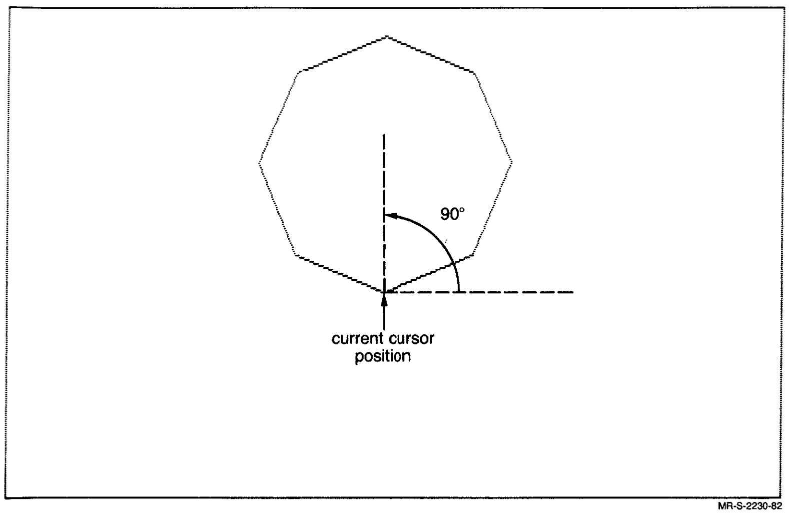
Figure 5–25: Picture Illustrating a POLYGC Call that Contains a Negative Angle



```
POLYGC (8, 125.0, 90.0)
```

draws an octagon with the vertices 125 units from the center of the octagon. The first vertex is at the current graphic cursor location and is located directly to the right of the center of the polygon (because of the 90–degree angle). After the octagon is drawn, the graphic cursor is again at the first vertex. See Figure 5–26.

Figure 5–26: Picture Illustrating a POLYGC Call that Contains a Positive Angle



RELATED SUBROUTINES

SLNPAT determines the line pattern.

SDGREE and SRADNS determine whether the angle is interpreted in degrees or radians (SRADNS is the default).

SSHADE shades the space enclosed by a non–rotated polygon if you specify the center of the polygon as the shade line. Some polygons cannot be shaded properly. See RESTRICTIONS.

POLYGN also draws a polygon, but its arguments are just the number of sides and the radius, not including the angle argument of POLYGC.

POLYGX also draws a polygon, but its arguments are the number of sides and the coordinates of one of the vertices.

BOX and RELBOX draw rectangles.

ARC, ARCC, CIRCC, CIRCLE, and CIRCXY draw arcs and circles.

RESTRICTIONS

See Section 2.5, Shading Picture Objects.

ERROR MESSAGES

1. Error Code 220

```
%RGL-W-INVSIDES, invalid number of sides
```

   means that you specified less than 3 as a value for the nsides argument. (More than 35 is, by RGL/11 definition, a circle).

PROGRAM EXAMPLE

none

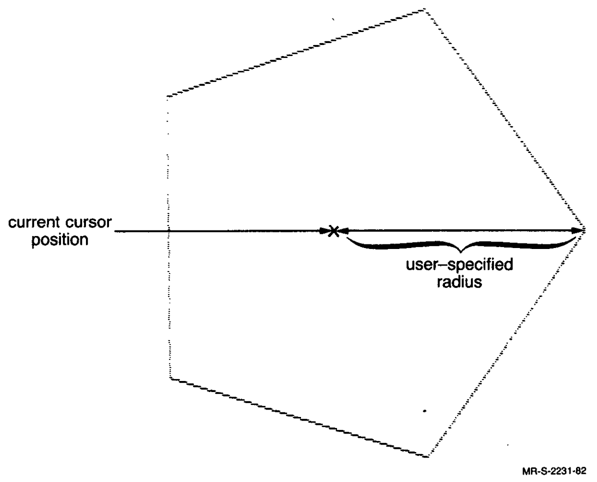
POLYGN

PURPOSE

POLYGN draws a regular polygon of the number of sides you specify and determines the size of the polygon from the radius you specify. The current graphic cursor location defines the center of the polygon. POLYGN draws in a counterclockwise direction, the first vertex being directly to the right of the center of the polygon. The polygon is symmetrical along the horizontal line that extends from the center.

After the polygon is drawn, the graphic cursor is again at its starting location, the center of the polygon. See Figure 5–27.

Figure 5–27: Picture Illustrating a POLYGN Call



FORM

POLYGN (nsides, radius)

ARGUMENT DESCRIPTION

nsides	is an integer expression that defines the number of sides of the polygon. The value for nsides must be greater than two.
radius	is a floating point expression that defines the distance in world coordinates from the center to any vertex of the polygon.

EXAMPLE

```
POLYGN (5, 200.0)
```

draws a pentagon whose center is at the current graphic cursor location and whose vertices are 200 units from the center of the pentagon. After the polygon is drawn, the graphic cursor is again at its starting location, the center of the pentagon.

RELATED SUBROUTINES

SLNPAT determines the line pattern.

SSHADE shades the space enclosed by a non–rotated polygon if you specify the center of the polygon as the shade line. There are certain polygons that cannot be shaded properly. See RESTRICTIONS.

POLYGC also draws a polygon, but its arguments are the number of sides, the radius, and the angle of the radius. The current graphic cursor location defines the location of a vertex, rather than the center of the polygon.

POLYGX also draws a polygon, but its arguments are the number of sides and the coordinates of one of the vertices.

BOX and RELBOX draw rectangles.

ARC, ARCC, CIRCC, CIRCLE, and CIRCXY draw arcs and circles.

RESTRICTIONS

See Section 2.5, Shading Picture Objects.

ERROR MESSAGES

1. Error Code 220

```
%RGL-W-INVSIDES, invalid number of sides
```

   means that you specified a value of less than three for the nsides argument. (More than 35 is, by RGL/11 definition, a circle.)

PROGRAM EXAMPLE

none

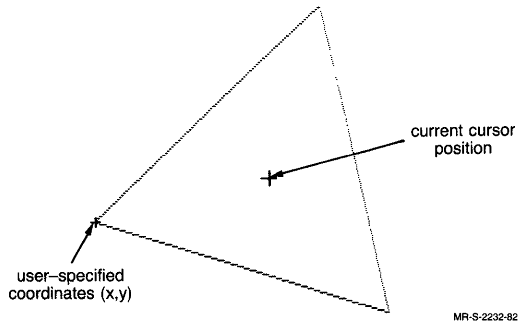
POLYGX

PURPOSE

POLYGX draws a regular polygon according to the number of sides and vertex location you specify. The current graphic cursor location defines the center of the polygon. POLYGX starts drawing at the vertex location you specify.

After the polygon is drawn, the graphic cursor is again at its starting location, the center of the polygon. See Figure 5–28.

Figure 5–28: Picture Illustrating a POLYGX Call



FORM

POLYGX (nsides, x, y)

ARGUMENT DESCRIPTION

nsides	is an integer expression that defines the number of sides of the polygon. The value for the argument must be greater than two.
x and y	are floating point expressions that define in world coordinates the location of one of the vertices of the polygon.

EXAMPLE

```
POLYGX (3, 300.0, 400.0)
```

draws an equilateral triangle with one vertex at location (300., 400.).

RELATED SUBROUTINES

SLNPAT determines the line pattern.

SSHADE shades the space enclosed by a non–rotated polygon if you specify the center of the polygon as the shade line. See RESTRICTIONS.

POLYGN also draws a polygon, but its arguments are the number of sides and the radius.

POLYGC also draws a polygon, but its arguments are the number of sides, the radius, and the angle of the radius, and it uses the current graphic cursor location as the location of a vertex, not the center, of the polygon.

BOX and RELBOX draw rectangles.

ARC, ARCC, CIRCC, CIRCLE, and CIRCXY draw arcs and circles.

RESTRICTIONS

See Section 2.5, Shading Picture Objects.

ERROR MESSAGES

1. Error Code 220

   ```
   %RGL-W-INVSIDES, invalid number of sides
   ```

   means that you specified a value for nsides that is less than three. (More than 35 is, by RGL/11 definition, a circle.)

PROGRAM EXAMPLE

none

POLYLN

PURPOSE

POLYLN (*Poly Line*) draws connected lines using data from the x and y arrays to define the endpoints of the lines. The current graphic cursor location is the first point on the line drawn by POLYLN.

After the lines are drawn, the graphic cursor is left at the last endpoint.

See Section 2.3, Drawing Graphic Objects from Data Arrays, for more information.

FORM

POLYLN (number, xarray, yarray)

ARGUMENT DESCRIPTION

number	is an integer expression that specifies the number of lines to draw. The value for this argument must be less than or equal to the number of elements in the smaller array.
xarray	is an array of floating point values that specify the x–axis coordinates for the lines to be drawn; xarray values use world coordinates.
yarray	is an array of floating point values that specify the y–axis coordinates for the lines to be drawn; yarray values use world coordinates.

EXAMPLE

```
POLYLN (20, XDATA, YDATA)
```

draws twenty lines. The x and y coordinates of the endpoints of the lines are in the arrays XDATA and YDATA.

RELATED SUBROUTINES

SLNPAT determines the line pattern.

RELPLN also draws connected lines from X and Y arrays, but it requires that the data specify the x– and y–distances from the current graphic cursor location.

LINE and RELLIN ~~also draw~~ connected lines, but these subroutines draw only one line each time they are called.

RESTRICTIONS

The number argument must be less than or equal to the number of elements in the smaller of the two arrays, xarray and yarray. If it is larger, the resultant graphic object is indeterminate.

ERROR MESSAGES

1. Error Code 340

   `%RGL-W-INVNUMEL, 'x' is an invalid number of elements`

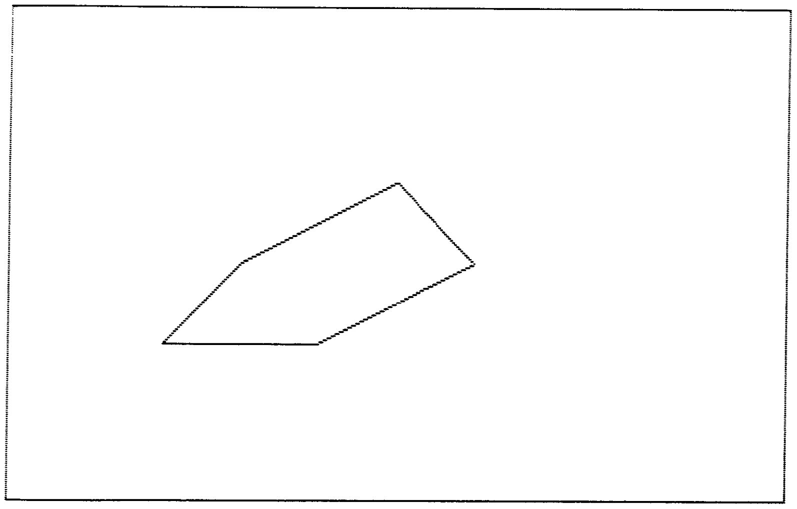
   means that you specified a value for the number argument that was zero or negative. POLYLN immediately returns control to the calling program.

PROGRAM EXAMPLE

The following program draws a 5–sided figure (see Figure 5–29).

```
      DIMENSION xdata (5), ydata (5)
      DATA xdata /600.,500.,300.,200.,400./
      DATA ydata /300.,400.,300.,200.,200./
C
C  Initialize VT125 and draw a box around the screen.
      CALL INITGR(5)
      CALL CLRSCR
      CALL CLRTXT
      CALL SWINDO (0.,0.,1000.,625.)
      CALL BOX (0.,0.,1000.,625.)
C
C  Move to the final x,y coords. in the XDATA and YDATA
C  arrays. This is done to close the figure drawn by POLYLN.
      CALL MOVE (400.,200.)
      CALL POLYLN (5,xdata,ydata)
C
      END
```

Figure 5–29: Picture Illustrating a POLYLN Call



PPOINT

PURPOSE

PPOINT (*Plot Point*) plots a single data point. It plots with respect to either linear or logarithmic graph paper, using one or more y–axes. It can use any marker you specify to mark the data point.

See Section 3.8.2, Interactive Data Plotting, for more information.

FORM

PPOINT (x, y, {'yaxis'}, {marker})

ARGUMENT DESCRIPTION

x and y	are floating–point expressions that specify the x and y coordinates of the point to be plotted.
'yaxis'	is an optional single–character string expression that specifies the y–axis PPOINT will use. Valid arguments are 'L' (for left y–axis) or 'R' (for right y–axis). Its default value is conditional on which y–axis (axes) is scaled:

- If only the right y–axis is scaled, the default is 'R' and the program continues.
- If only the left y–axis is scaled, the default is 'L' and the program continues.
- If both left and right are scaled, the default is 'L' and the program continues.
- If neither one is scaled, the default is 'L,' error 650 (axis not scaled) is invoked, and control returns to the calling program.

marker	is an optional integer expression that specifies the marker to be used to display the point. There are 11 markers, 0 through 10; refer to the marker code listed in the MARKER subroutine for the list of markers. The default is 0, a dot.

EXAMPLE

```
PPOINT (3.2,5.4,'R',1)
```

plots a point marked by a square at x–coordinate 3.2, y–coordinate 5.4. The y–coordinate refers to the right y–axis.

RELATED SUBROUTINES

DPAPER creates the grid to plot data on.

LNAXIS assigns a set of numeric values to the cell divisions of an axis and labels the cell divisions with those values. It also gives the entire axis a label that identifies its purpose.

LTAXIS assigns a set of numeric values to the cell divisions of a graph's top or bottom x–axis or left or right y–axis, and displays a text label at each cell boundary. It also labels the axis as a whole.

PDATA maps data arrays into rectangular coordinates and plots them on the graph paper you specify.

SCOLOR controls the drawing color PPOINT uses.

RESTRICTIONS

You must scale the x– and y–axes before you call PPOINT. You can do this either (1) by calling DPAPER and then either LTAXIS or LNAXIS, or (2) by calling PDATA.

If you want to change the drawing color PPOINT will use for the marker, you must precede a call to PPOINT with a SCOLOR call. You can use only color numbers 2 and 3 for setting the color. Color numbers 0 and 1 are used for the screen color and the grid lines, respectively. If you use 0 or 1, you alter the screen or grid colors.

ERROR MESSAGES

1. Error Code 270

   ```
   %RGL-W-BADMARKER, 'x' is not a valid marker
   ```

   means that you specified a value for the marker argument that was not from 0 through 10. When this error occurs, the argument assumes the default value and the program continues.

2. Error Code 600

   ```
   %RGL-W-AID, 'x' is an invalid axis identifier
   ```

   means that you specified a value for the yaxis argument that was not "L" or "R," or that did not match the axisid argument of the LNAXIS subroutine. When this error occurs, the argument assumes the default value and the program continues.

3. Error Code 640

   ```
   %RGL-W-XNS, X axis has not been scaled
   ```

   means that you established the type of graph with a call to DPAPER but did not call LNAXIS, LTAXIS, or PDATA to scale the x–axis. When this error occurs, PPOINT returns control to the calling program.

4. Error Code 650

   ```
   %RGL-W-YNS, Y axis has not been scaled
   ```

   means that you called DPAPER but you did not call either LNAXIS or PDATA to scale the y–axis. When this error occurs, PPOINT returns control to the calling program.

5. Error Code 740

   ```
   %RGL-W-NPL, logarithm of non-positive number
   ```

   means the value for either the x or y argument was non–positive and was to be used on a logarithmic axis. When this error occurs, it does not interrupt the data plotting sequence. However, the point will not be plotted.

6. Error Code 750

   ```
   %RGL-W-NOPAPER, no graph paper has been drawn
   ```

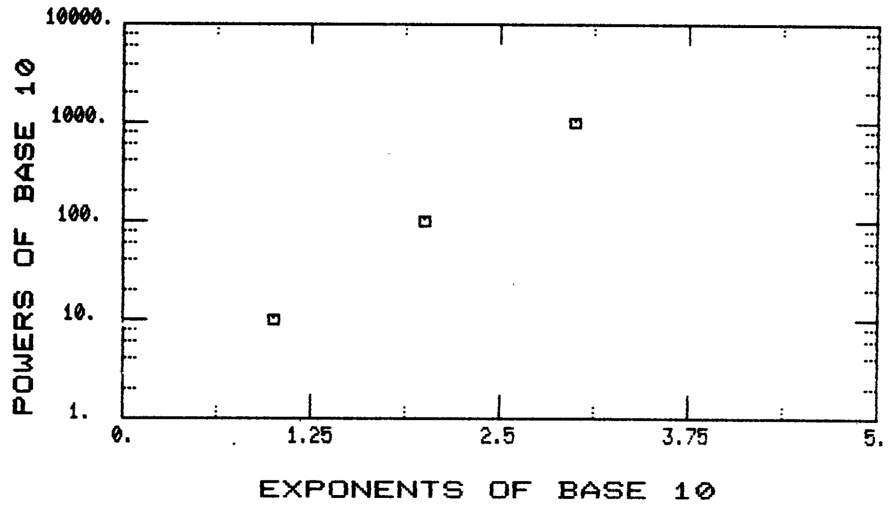
   means that you did not call DPAPER or PDATA before calling PPOINT. The call is ignored and control returns to the calling program.

PROGRAM EXAMPLE

The following program fragment creates a piece of 4–cycle semi–log graph paper and plots three data points on it. See Figure 5–30. Other examples of PPOINT calls are in the LFREE and LFIXED Program Example sections.

```
CALL DPAPER ('LIN',4,2,'LOG',4,5,'GRAY2')
CALL LNAXIS ('XB','EXPONENTS OF BASE 10',0.,5.,,TRUE.)
CALL LNAXIS ('YL','POWERS OF BASE 10',1.,1000.,,TRUE.)
CALL PPOINT (1.,10.,'L',1)
CALL PPOINT (2.,100.,'L',1)
CALL PPOINT (3.,1000.,'L',1)
```

Figure 5–30: Graph Illustrating PPOINT Calls



PURPOSE

RELBOX (*Rel*ative *Box*) draws a box according to the width and height you specify and beginning at the current graphic cursor location. You can determine the orientation of the box in relation to the current graphic cursor location by specifying positive or negative values for the width and height.

After the box is drawn, the graphic cursor is again at its original location.

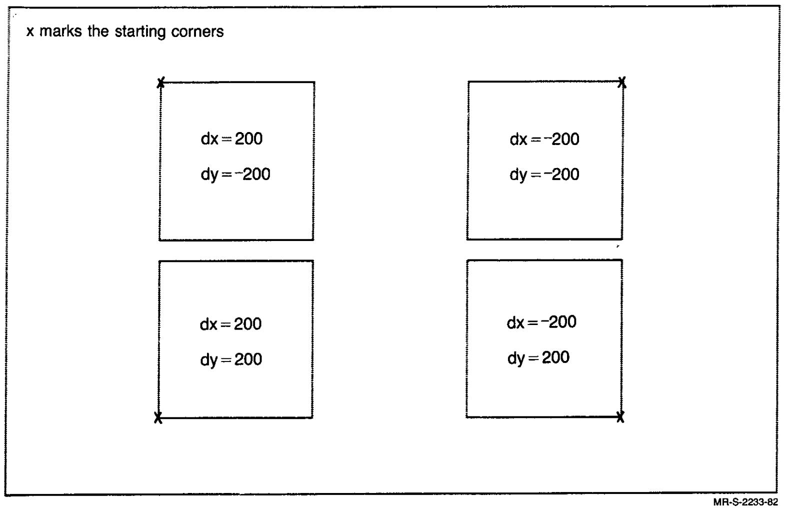
FORM

RELBOX (dx, dy)

ARGUMENT DESCRIPTION

dx and dy	are floating point expressions; dx, the width of the box, is the distance along the x–axis from the current graphic cursor location, and dy, the height, is the distance along the y–axis. These arguments use world coordinates; they can be positive or negative. The sign of each value determines the starting corner of the box. When dx is a positive number, the line is drawn to the right of the starting location; when the width is a negative number, the line is drawn to the left. When the height, dy, is a positive number, the line is drawn upward from the end of the dx line; when it is negative, the line is drawn downward. Thus, given any cursor location, there are four different directions from which you can draw the box. Figure 5–31 shows the possible starting corners of the boxes. The SWINDO subroutine was first called with the following arguments: (0.,0.,1000.,625.).

Figure 5–31: Illustration of RELBOX Calls



EXAMPLE

```
RELBOX (-100.0, 200.0)
```

draws a box that has the current graphic cursor location as the lower right corner of the box, and leaves it there after completing the drawing. The box is 100 units wide and 200 units high.

RELATED SUBROUTINES

SLNPAT determines the line pattern.

BOX also draws a box, but its arguments are the coordinates for two diagonally opposite corners of the box.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

The following program fragment sets the window, moves the graphic cursor to location (500., 500.), and draws a box that has that location as its upper right corner and has sides 200 units long. After the box is drawn, the graphic cursor is left at location (500., 500.).

```
CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
CALL MOVE (500.0, 500.0)
CALL RELBOX (-200.0, -200.0)
```

PURPOSE

RELLIN (*Rel*ative *Line*) draws a line from the current graphic cursor location to a point determined by the x– and y–distances you supply.

RELLIN leaves the graphic cursor at the end of the line it draws.

FORM

RELLIN (dx, dy)

ARGUMENT DESCRIPTION

dx and dy	are floating point expressions that define the x– and y–distances in world coordinates from the current graphic cursor location to the endpoint of the line.

EXAMPLE

```
RELLIN (200.0, 125.0)
```

draws a line from the current graphic cursor location to a location that is 200 units away in the x–direction and 125 units away in the y–direction.

RELATED SUBROUTINES

SLNPAT determines the line pattern.

LINE also draws a line, but its arguments are the x and y coordinates of the endpoint of the line, rather than the x– and y–distance from the current graphic cursor location.

RESTRICTIONS

none

ERROR MESSAGES

none

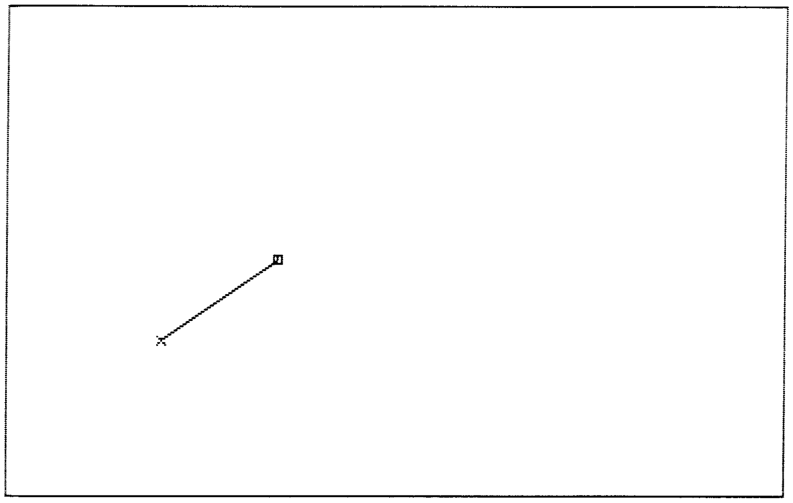
PROGRAM EXAMPLE

The following program sets the window, moves the graphic cursor to location (200., 200.), and draws a line that is 150 units in the x–direction and 100 units in the y–direction. The graphic cursor is left at location (350., 300.). Markers are placed at the beginning and end of the line. See Figure 5–32.

```
C  Initialize VT125 and draw a box around the screen.
      CALL INITGR (5)
      CALL CLRSCR
      CALL CLRTXT
      CALL SWINDO (0.,0.,1000.,625.)
      CALL BOX (0.,0.,1000.,625.)
C
C  Move to the start of the line and draw an 'X'
      CALL MOVE (200.,200.)
      CALL MARKER (5,,)
C
```

```
C  Now call RELLIN to draw a line and then draw a
C  box at the final cursor location.
        CALL RELLIN (150.,100.)
        CALL MARKER (1,350.,300.)
C
        END
```

Figure 5–32: Illustration of a RELLIN Call



RELMKR

PURPOSE

RELMKR (*Rel*ative *M*ar*k*e*r*) displays a marker at the distance you specify from the current graphic cursor location. The location you specify is where the marker's center is displayed. The default marker is a dot, marker number 0.

After the marker is displayed, RELMKR leaves the graphic cursor at the center of the marker.

FORM

RELMKR ({number}, dx, dy)

ARGUMENT DESCRIPTION

number	is an optional argument; it is an integer expression that identifies the marker you want to display. If you do not specify a number, RELMKR uses the default marker, number 0, a dot. There are eleven markers, whose numbers range from 0 to 10. They are the same as listed in Table 5–3 in the MARKER subroutine:

0	—	point
1	—	square
2	—	octagon
3	—	triangle
4	—	cross
5	—	X
6	—	Y
7	—	diamond
8	—	arrowhead
9	—	hourglass
10	—	point within a circle

dx and dy	are floating point expressions that define the x and y distances in world coordinates from the current graphic cursor location to the location where the marker is to be displayed.

EXAMPLE

```
RELMKR (2, 10.0, 20.0)
```

displays an octagon that is 10 units in the x–direction and 20 units in the y–direction from the current graphic cursor location. The graphic cursor is left at the specified location.

RELATED SUBROUTINES

MARKER also displays a marker, but it displays it either at the current graphic cursor location or at an absolute location you specify.

PPOINT plots individual points on a graph using a specified marker.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 270

   ```
   %RGL-W-BADMARKER, % is not a valid marker
   ```

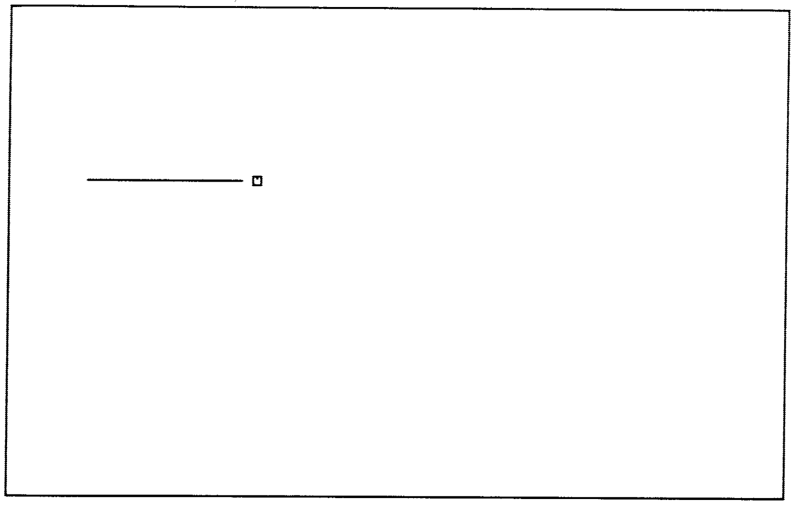
   means that you specified a marker number that is less than 0 or more than 10. RELMKR defaults to a marker number of 0, a dot, and continues.

PROGRAM EXAMPLE

The following program draws a line across the screen, then places a marker 20 units away in the x direction from the end of the line. See Figure 5–33.

```
C   Initialize VT125 and draw a box around the screen.
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
          CALL SWINDO (0.,0.,1000.0.,625.)
          CALL BOX (0.,0.,1000.,625.)
C
C   Draw a line from 100.,400 to 300.,400. then draw
C   a box twenty world coordinates away from the
C   line in the x direction using RELMKR.
          CALL MOVE (100.,400.)
          CALL LINE (300.,400.)
          CALL RELMKR (1,20.,0.)
C
          END
```

Figure 5–33: Picture Illustrating a RELMKR Call



RELMOV

PURPOSE

RELMOV (*Rel*ative *Mov*e) moves the graphic cursor by the specified x– and y–distances from its current location. RELMOV does not draw a graphic object of any kind.

FORM

RELMOV (dx, dy)

ARGUMENT DESCRIPTION

dx and dy	are floating point expressions that define the distance in world coordinates for the graphic cursor to move from its current location. You can use either positive or negative values.

EXAMPLE

```
RELMOV (100.0, 150.0)
```

moves the graphic cursor 100 units in the x–direction and 150 units in the y–direction from its current location.

RELATED SUBROUTINES

MOVE also moves the graphic cursor, but to an absolute location.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

RELPLN

PURPOSE

RELPLN (*Rel*ative *P*oly *Line*) draws connected lines using data from dxarrays and dyarrays to define the x– and y–distances of the lines. The current location of the graphic cursor is the location of the first point RELPLN draws from.

RELPLN leaves the graphic cursor at the end of the last line.

FORM

RELPLN (number, dxarray, dyarray)

ARGUMENT DESCRIPTION

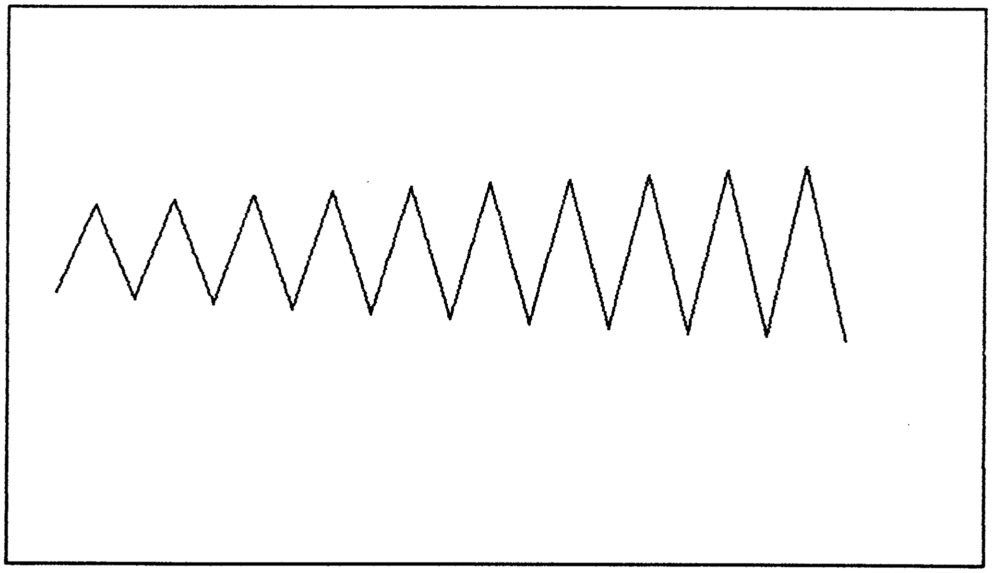
number	is an integer expression that specifies the number of lines to draw. This argument must be less than or equal to the number of elements in the smaller array.
dxarray	is an array of floating point values that specify the x–distances of the line lengths in world coordinates.
dyarray	is an array of floating point values that specify the y–distances of the line lengths in world coordinates.

EXAMPLE

```
RELPLN (20, DXDATA, DYDATA)
```

draws 20 lines. The x and y distances of the lines are defined in arrays called DXDATA and DYDATA. See Figure 5–34.

Figure 5–34: Picture Illustrating a RELPLN Call



RELATED SUBROUTINES

SLNPAT determines the line pattern.

POLYLN also draws connected lines using coordinates in xarrays and yarrays, but the data specifies the endpoints of the lines instead of the lengths.

LINE and RELLIN draw lines, but they draw only one line each time they are called.

RESTRICTIONS

The number argument must be less than or equal to the number of elements in the smaller array. If it is larger, the resultant graphic object is indeterminate.

ERROR MESSAGES

1. Error Code 340

   ```
   %RGL-W-INVNUMEL, 'x' is an invalid number of elements
   ```

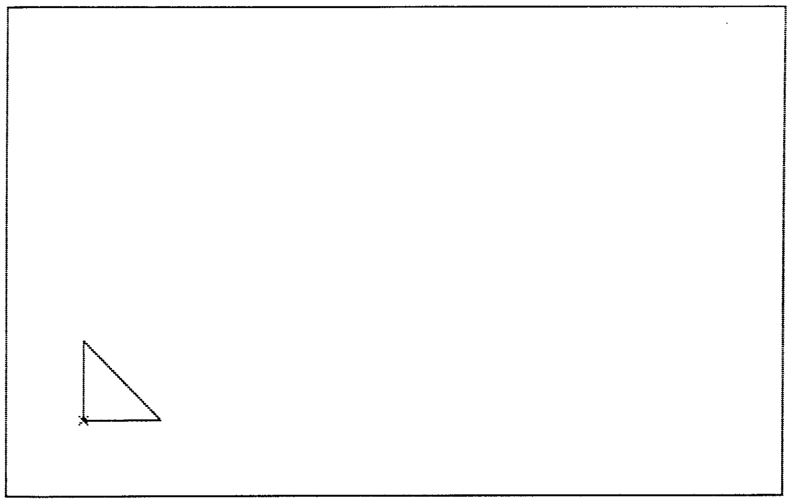
   means that you specified a value for the number argument that was zero or negative. RELPLN immediately returns control to the calling program.

PROGRAM EXAMPLE

The following program sets the window, moves the graphic cursor to where the graphic object will start, and draws a triangle. The graphic cursor is left at location (100.,100.). See Figure 5–35.

```
      DIMENSION dxdata (3), dydata (3)
C
C  Notice that x coordinates and the y coordinates
C  in DXDATA and DYDATA add up to zero. This ensures
C  a closed figure.
      DATA dxdata /100.,-100.,0./
      DATA dydata /0.,100.,-100./
C
C  Initialize VT125 and draw a box around the screen.
      CALL INITGR (5)
      CALL CLRSCR
      CALL CLRTXT
      CALL SWINDO (0.,0.,1000.,625.)
      CALL BOX (0.,0.,1000.,625.)
C
C  Move to where we want one of the triangle's vertices,
C  draw an 'X' then use RELPLN to draw the triangle.
      CALL MOVE (100.,100.)
      CALL MARKER (5,,)
      CALL RELPLN (3,dxdata,dydata)
C
      END
```

Figure 5–35: Picture Illustrating a RELPLN Call

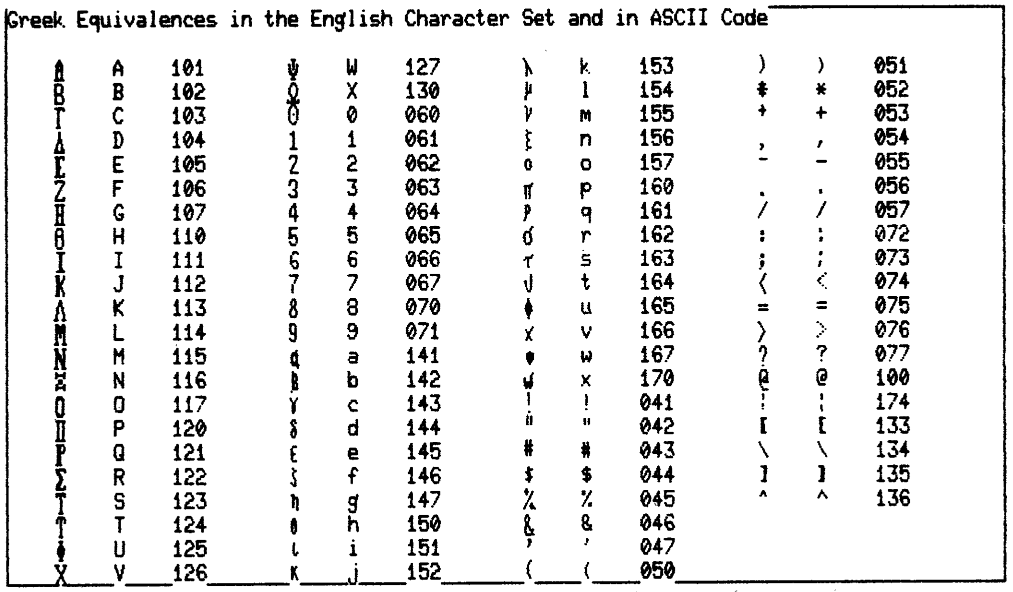


PURPOSE

SCHRST (*Se*t *Cha*racter *Set*) invokes a character set for writing all subsequent graphic text, that is, for text generated by TEXT, LINETX, LNAXIS, LTAXIS, and PDATA calls. In addition to the standard character set resident in your terminal, RGL/11 provides a Greek character set on your distribution volume, in a file named GREEK.FNT.

To access the characters in GREEK.FNT, you can use the corresponding English character or the ASCII code for that character. Table 5–2 lists the characters in GREEK.FNT and their equivalences.

Table 5–2: Greek Equivalences in the English Character Set and in ASCII Code



Greek Equivalences in the English Character Set and in ASCII Code

Α	A	101	Ψ	W	127	λ	k	153	)	)	051
Β	B	102	Ω	X	130	μ	l	154	*	*	052
Γ	C	103	0	0	060	ν	m	155	+	+	053
Δ	D	104	1	1	061	ξ	n	156	,	,	054
Ε	E	105	2	2	062	ο	o	157	-	-	055
Ζ	F	106	3	3	063	π	p	160	.	.	056
Η	G	107	4	4	064	ρ	q	161	/	/	057
Θ	H	110	5	5	065	σ	r	162	:	:	072
Ι	I	111	6	6	066	τ	s	163	;	;	073
Κ	J	112	7	7	067	υ	t	164	<	<	074
Λ	K	113	8	8	070	φ	u	165	=	=	075
Μ	L	114	9	9	071	χ	v	166	>	>	076
Ν	M	115	α	a	141	ψ	w	167	?	?	077
Ξ	N	116	β	b	142	ω	x	170	@	@	100
Ο	O	117	γ	c	143	!	!	041	|	|	174
Π	P	120	δ	d	144	"	"	042	[	[	133
Ρ	Q	121	ε	e	145	#	#	043	\	\	134
Σ	R	122	ζ	f	146	$	$	044	]	]	135
Τ	S	123	η	g	147	%	%	045	^	^	136
Υ	T	124	θ	h	150	&	&	046			
Φ	U	125	ι	i	151	'	'	047			
Χ	V	126	κ	j	152	(	(	050			

The standard set in the terminal is invoked by the INITGR call, which should be the first call in every graphic program, or by the SCHRST call. Since the standard set always resides in the terminal, it does not need to be loaded. However, to invoke any other character set, you must first load it with a call to LCHRST before calling SCHRST to make it the current writing text.

FORM

SCHRST (number)

ARGUMENT DESCRIPTION

number	is an integer expression that specifies the number of the character set to be used: 0, the standard English character set, or 1, for the alternate character set.

EXAMPLE

```
SCHRST (0)
```

invokes character set 0, the standard character set. Any graphic text displayed on the screen after this call is written in the standard character set.

```
SCHRST (1)
```

invokes the alternate character set previously loaded by a LCHRST call. Any graphic text displayed on the screen after this call is written in that character set.

RELATED SUBROUTINES

LCHRST loads the character definitions from a file into the alternate character set in the terminal's memory.

LINETX, TEXT, LNAXIS, LTAXIS, PDATA, and SSHADE are the subroutines that use the character set that LCHRST loads and SCHRST invokes.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 150

   ```
   %RGL-W-INVCHARSET, number is not a valid character set
   ```

   means that the character set number you specified is less than 0 or greater than 1. SCHRST supplies 0, the default, and the program continues.

2. Error Code 330

   ```
   %RGL-W-NOCHARSET, character set 'x' has not been loaded
   ```

   means that you are trying to invoke a character set that has not been loaded. RGL/11 assumes you wanted to invoke a character set that was loaded previously. The result will depend on your programs. If a character set was loaded in a previous program, RGL/11 uses that set and continues. If one was not, RGL/11 does not load one; therefore there will be no character set to use, blank text will be printed.

PROGRAM EXAMPLE

The following program fragment loads the Greek character set included on your distribution volume into alternate character set 1 in the terminal's memory using the current default LUN, and causes five characters of the set to be displayed.

```
CALL LCHRST (1, 'GREEK.FNT', )
CALL SCHRST (1)
CALL TEXT ('abcde')
```

SCOLOR

PURPOSE

SCOLOR (*Set Color*) associates a color name with a color number and establishes the current drawing color number. Color number 0 controls the screen color; color numbers 1, 2, and 3 control the current drawing color. Color names GRAY0, GRAY1, GRAY2 or GRAY3 can be associated with any color number. These four colors appear on a color monitor as black, blue, red, and green respectively.

If you specify color number 1, 2, or 3, and any color name, all objects you subsequently draw are displayed in that color, and all objects previously written with that color number are now displayed in the color you specified, regardless of their previous color.

If you specify color number 0 and any color name, the screen color immediately changes to the color you specified, and the drawing color is also that color name (so objects would appear invisible unless you immediately change the drawing color). It is not recommended to change the screen color until you are completely familiar with the SCOLOR and SWMODE subroutines.

When you follow recommended procedure and call INITGR at the beginning of each program, INITGR makes the following assignments for the VT125 screen. INITGR makes the screen color GRAY0, color number 1, GRAY1, 2, GRAY2, and 3, GRAY3. It also enables color number 3 as the current drawing color. If you do not call INITGR, the screen and drawing colors are indeterminate.

See Section 2.10, Selecting Gray Shades or Color, for more information.

FORM

SCOLOR ({'name'}, number)

ARGUMENT DESCRIPTION

name — is an optional string expression that defines the color name. Valid names are GRAY0, GRAY1, GRAY2, and GRAY3. They are displayed on black–and–white and color monitors as follows:

Valid Names	Black–and–White Monitor	Color Monitor
GRAY0	black	black
GRAY1	dark gray	blue
GRAY2	light gray	red
GRAY3	white	green

If you do not specify a name, the color name last associated with the color number is retained.

number	is an integer expression that represents a logical color number. The number argument indicates which number in the terminal's memory is to be loaded and used as the drawing color number for subsequent calls.

When you associate a new color name with a color number used in previous drawings, all objects drawn with that color number change to the new color name now associated with it. For example, if a box was drawn in GRAY1 when it was paired with number 2, a new SCOLOR call that paired GRAY3 with number 2 would change the box to GRAY3.

An argument of 0 affects both the background color and the drawing color; you should usually restrict the argument to 1, 2, or 3.

EXAMPLE

```
CALL SCOLOR ('GRAY2', 1)
```

associates GRAY2 with logical color 1 and immediately changes all graphic objects that were drawn with color number 1 to the shade GRAY2. In addition, the writing color for subsequent drawing is set to GRAY2. If you have a color monitor, the objects are displayed in red.

RELATED SUBROUTINES

Data plotting subroutines have arguments to define the drawing color; they are DPAPER, PDATA, and PPOINT.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 010

```
%RGL-W-CNF, color 'x' not found
```

means that you misspelled the color name or referenced a non–existent color name. SCOLOR defaults to the current drawing color, associates that color with the color number specified, and the program continues.

PROGRAM EXAMPLE

The following program fragment sets the window, sets the drawing color, and then draws a line.

```
CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
CALL SCOLOR ('GRAY3', 1)
CALL MOVE (100.0, 100.0)
CALL LINE (250.0, 300.0)
```

PURPOSE

SDEBUG (*Set Debug*) directs the RGL/11 subroutines to display error messages on the terminal. It is the initial default, so you need to call it only if you want to cancel a previous call to SNDBUG.

FORM

SDEBUG

ARGUMENT DESCRIPTION

none

EXAMPLE

```
SDEBUG
```

RELATED SUBROUTINES

SNDBUG prevents RGL/11 subroutines from displaying error messages.

GETSTA returns a value, an error code, in its istat argument whether or not an error message is displayed on the screen. You can read this value any time you want to see if an error occurred. For example, you may want to call GETSTA if the program is operating in a set–no–debug mode, after a subroutine call, to see what the last error was. After the istat argument has been read, GETSTA returns it to "001, the code for an error–free condition.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

SDGREE

PURPOSE

SDGREE (*Set Degree*) causes angle arguments to be interpreted in degrees, rather than radians. The initial default is to treat angle arguments as radians.

FORM

SDGREE

ARGUMENT DESCRIPTION

none

EXAMPLE

SDGREE

RELATED SUBROUTINES

SRADNS causes angle arguments to be interpreted in radians.

ARC, ARCC, CIRCC, and POLYGC take angle arguments.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

PURPOSE

SLNPAT (*S*et *L*i*n*e *Pat*tern) sets the pattern for lines in any subsequent picture objects. There are ten patterns. You can specify a multiplier with each one to lengthen the pattern.

You can change the line pattern as often as you want by making repeated calls to SLNPAT.

The initial default for SLNPAT is 1, a solid line.

See Section 2.4, Changing the Line Pattern, for more information.

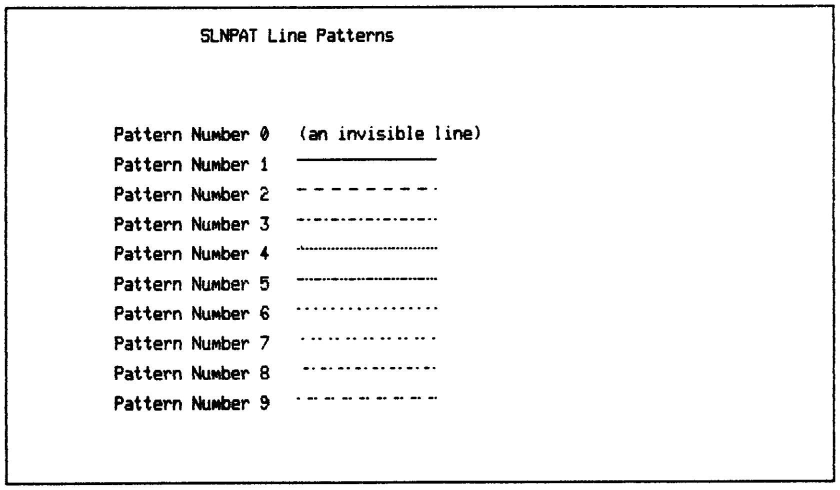
FORM

SLNPAT ({number}, {mult})

ARGUMENT DESCRIPTION

number — is an optional integer expression that defines the line pattern. Valid line pattern numbers range from 0 through 9. The default is 1, a solid line. The numbers and the patterns they represent are:

Table 5–4: Line Patterns and Their Identifying Numbers



```
SLNPAT Line Patterns

Pattern Number 0  (an invisible line)
Pattern Number 1
Pattern Number 2
Pattern Number 3
Pattern Number 4
Pattern Number 5
Pattern Number 6
Pattern Number 7
Pattern Number 8
Pattern Number 9
```

mult — is an optional argument; it is an integer expression that defines the pattern multiplier. Valid values range from 2 through 9. The default value is 2.

EXAMPLE

```
SLNPAT (3, 4)
```

draws subsequent picture objects in line pattern 3, a dot–dash line. Each element — the dot, dash, and space between — is enlarged to four times its regular size before the pattern is repeated.

```
SLNPAT (1, )
```

draws picture objects in line pattern 1, a solid line. The pattern multiplier is 2 by default.

RELATED SUBROUTINES

SSHADE shades with the current line pattern or a shade character.

ARC, ARCC, BOX, CIRCC, CIRCLE, CIRCXY, LINE, POLYGC, POLYGN, POLYGX, POLYLN, RELBOX, RELLIN, RELPLN, use the line pattern.

Among the data plotting subroutines, PDATA has an argument that defines the line pattern, using the same numbers as in the Line Pattern Chart.

RESTRICTIONS

none

ERROR MESSAGES

1. Error Code 200

   ```
   %RGL-W-INVLINE, 'x' is not a valid line pattern
   ```

   means that you specified a line pattern that is outside the range of 0 through 9 or a pattern multiplier outside the range of 2 through 9. RGL/11 defaults to a line pattern of 1 and a multiplier of 2 and continues.

PROGRAM EXAMPLE

The following program fragment sets the window, sets the line pattern to pattern 2 (a pattern of dashes), and draws a line from location (200., 200.) to location (100., 100.).

```
CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
CALL SLNPAT (2, )
CALL MOVE (200.0, 200.0)
CALL LINE (100.0, 100.0)
```

SNDBUG

PURPOSE

SNDBUG (*Set Nodebug*) prevents RGL/11 subroutines from displaying error messages on the terminal. SDEBUG is the initial default, so you need to call SNDBUG only if you want to stop the displaying of error messages.

FORM

SNDBUG

ARGUMENT DESCRIPTION

none

EXAMPLE

```
SNDBUG
```

RELATED SUBROUTINES

SDEBUG directs the RGL/11 subroutines to display error messages on the terminal.

GETSTA returns a value, an error code, in its istat argument, whether or not error messages are displayed on the screen. You can read this value any time you want to see if an error occurred during the time frame you are interested in. For example, you may want to call GETSTA after a subroutine call to see what the state of the last error was. After the istat argument has been returned, GETSTA returns it to "001, the code for an error–free condition.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

SNSHAD

PURPOSE

SNSHAD (*S*et *N*o *Shad*e) turns off the shading that you specified with a call to SSHADE. The initial default is to have shading turned off, so you do not have to call SNSHAD unless you want to cancel a call to SSHADE.

FORM

SNSHAD

ARGUMENT DESCRIPTION

none

EXAMPLE

```
SNSHAD
```

RELATED SUBROUTINES

SSHADE lets you shade picture objects.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

SRADNS

PURPOSE

SRADNS (*S*et *Rad*ia*ns*) causes angle arguments to be interpreted in radians rather than degrees. The initial default is to treat angle arguments as radians, so you need to call SRADNS only if you want to cancel a call to SDGREE.

FORM

SRADNS

ARGUMENT DESCRIPTION

none

EXAMPLE

```
SRADNS
```

RELATED SUBROUTINES

SDGREE causes the angle arguments to be interpreted as degrees.

ARC, ARCC, CIRCC, and POLYGC take angle arguments.

RESTRICTIONS

none

ERROR MESSAGES

none

PROGRAM EXAMPLE

none

SSHADE

PURPOSE

SSHADE (*S*et *Shade*) shades the space within subsequent picture objects to the horizontal shade line you specify. SSHADE uses either the line pattern or character that you choose. If you want to shade a circle, arc, or polygon, you must specify the line through the center as the shade line. There are some figures that cannot be "filled in" by the shading function since SSHADE does not use a "fill" algorithm; rather, it shades to a shade line.

Shading stays in effect until you call SNSHAD or INITGR. The default condition is no shading.

See Section 2.5, Shading Picture Objects, for more information.

FORM

SSHADE ({yline}, {ipat})

ARGUMENT DESCRIPTION

yline	is an optional argument; it is a floating point expression that defines the horizontal shade line in world coordinates. All picture objects shade to this line. If you do not specify yline, the y–coordinate of the current graphic cursor location is used as the default.
ipat	is an optional argument; it is a byte expression that indicates the line pattern or character to shade with. Valid octal numbers are "1, and "40 through "176 (octal). If the value is "40 through "176, ipat uses the character that corresponds to that ASCII value. If you do not specify a value or if you give ipat a value of 1, SSHADE uses the current line pattern as the default.

EXAMPLE

```
SSHADE (200.0, "100)
SSHADE (200.0, '@')
```

These examples are equivalent; they shade all subsequent picture objects to the yline of 200 using the character whose ASCII value is an octal 100 (the @ character in the standard character set).

```
SSHADE (400.0,)
```

shades all subsequent picture objects to the yline of 400, using the current line pattern as the default.

RELATED SUBROUTINES

SLNPAT determines the current line pattern.

LCHRST and SCHRST determine the character set you shade with.

SNSHAD turns off shading.

RESTRICTIONS

See Section 2.5, Shading Picture Objects.

ERROR MESSAGES

1. Error Code 160

```
%RGL-W-INVSHADPAT, ipat is not a valid shading pattern
```

means that you specified an octal number that is (1) less than or equal to zero, (2) between "2 to "37, or (3) equal to or greater than "177. SSHADE supplies the default value (the current line pattern) and the program continues.

PROGRAM EXAMPLE

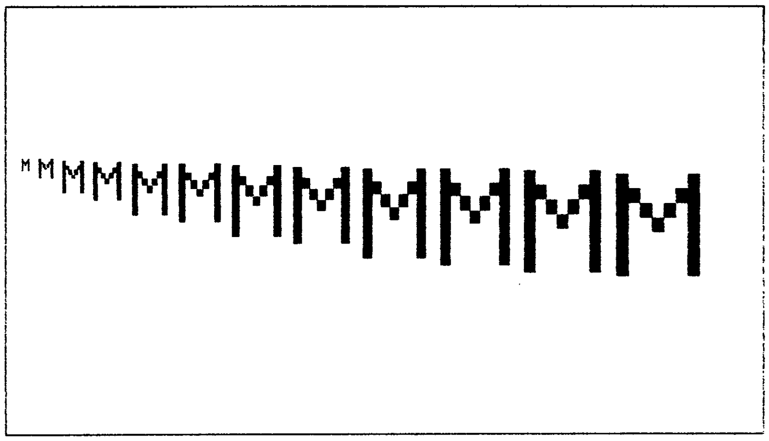
See Section 2.5, Shading Picture Objects, for programming examples.

STXSIZ

PURPOSE

STXSIZ (*S*et *Te*xt *Siz*e) lets you define the size of the graphic characters that TEXT displays. After you call STXSIZ, all messages that you write by calling TEXT are written at the specified size, until you call STXSIZ again and give it a specification of 1, standard size, or until an INITGR call. You can draw characters up to and including 16 times standard size. The first twelve character sizes are illustrated in Figure 5–36.

Figure 5–36: Sample STXSIZ Character Sizes



FORM

STXSIZ ({iwide}, {ihigh})

ARGUMENT DESCRIPTION

iwide	is an optional integer expression that defines the scale for the width of the graphic characters. Its value is from 1 through 16; its default is 1.
ihigh	is an optional argument; it is an integer expression that defines the scale for the height of the graphic characters. Its value is from 1 through 16. If you do not specify a value for ihigh, it becomes the integer nearest to 1.5 times the width or 16, whichever is less.

EXAMPLE

```
STXSIZ (2, )
```

makes all subsequent graphic text twice the standard size.

```
STXSIZ (1, )
```

returns all subsequent graphic text to the standard size.

RELATED SUBROUTINES

SCHRST determines the character set for writing graphics messages.

INITGR returns the character size to its initial default.

RESTRICTIONS

STXSIZ does not affect the size of graphic text that is written with a call to DPAPER, LINETX, LNAXIS, LTAXIS, or PDATA.

ERROR MESSAGES

1. Error Code 170

   ```
   %RGL-W-INVTEXSIZ, 'x' is not a valid text size
   ```

   means that either iwide or ihigh is less than 0 or more than 16. If you gave ihigh an invalid value, RGL/11 sets ihigh to 1.5 times iwide and continues. Also if you defaulted the iwide parameter or gave it an invalid value, RGL/11 sets iwide to 1 and ihigh to 2 and continues.

PROGRAM EXAMPLE

The following program fragment sets the window, writes a message at standard size, sets the character size to twice that size, and writes a second message.

```
CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
CALL MOVE (100.0, 125.0)
CALL TEXT ('Regular size characters')
CALL STXSIZ (2, )
CALL MOVE (100.0, 200.0)
CALL TEXT ('Double size characters')
```

Also see the program LABEL1 in Section 2.8, Labeling the Picture.

SWINDO

PURPOSE

SWINDO (*S*et *Windo*w) defines the window. The window is that portion of your coordinate system that is displayed on the graphic screen. The coordinates that you establish by this subroutine are called "world coordinates." The initial default condition sets the coordinates to be the same as the physical screen coordinates, and sets the origin location at the lower left of the graphic screen.

To maintain the same aspect ratio as the screen coordinates, use an 8 to 5 ratio (for example, 1000 units on the x–axis and 625 on the y–axis). When you use the screen's aspect ratio, units on the x–axis and the y–axis measure equal distances, and the ARC and CIRCLE commands create circles, not ellipses.

See Section 2.2, Defining Coordinate Systems, for more information on how to use SWINDO.

FORM

SWINDO (xleft, ybot, xright, ytop)

ARGUMENT DESCRIPTION

xleft, ybot, xright, ytop	are floating point expressions that define the edges of the window in world coordinates:

- xleft is the left edge of the window.
- ybot is the bottom of the window.
- xright is the right edge of the window.
- ytop is the top of the window.

EXAMPLE

```
SWINDO (0.0, 0.0, 1000.0, 625.0)
```

establishes a window that retains the aspect ratio and puts the origin in the lower left corner.

RELATED SUBROUTINES

INITGR resets the window coordinates to their initial default values.

RESTRICTIONS

The width and height of your window should be proportionate to the aspect ratio of the screen. Otherwise your picture will be distorted.

If you use SWINDO to window in on a very small part of a picture, unexpected results may occur. To avoid this problem, do not make the x–axis of the new window any smaller than one–twentieth of the x–axis of the original window, and do not make the y–axis any smaller than one–thirtieth of the original y–axis. Suppose you draw a picture in which there is a box in the lower left corner and a box in the upper right corner. Then you reduce the size of the window beyond the ratios mentioned above and redraw the entire image, including those elements outside the window. You will be requiring the graphic cursor to make such large moves that it will exceed the capabilities of the terminal. The picture will "wrap around" and may draw lines in unexpected locations.

ERROR MESSAGES

1. Error Code 100
   `%RGL-W-INVWINCOORD, invalid window coordinates`

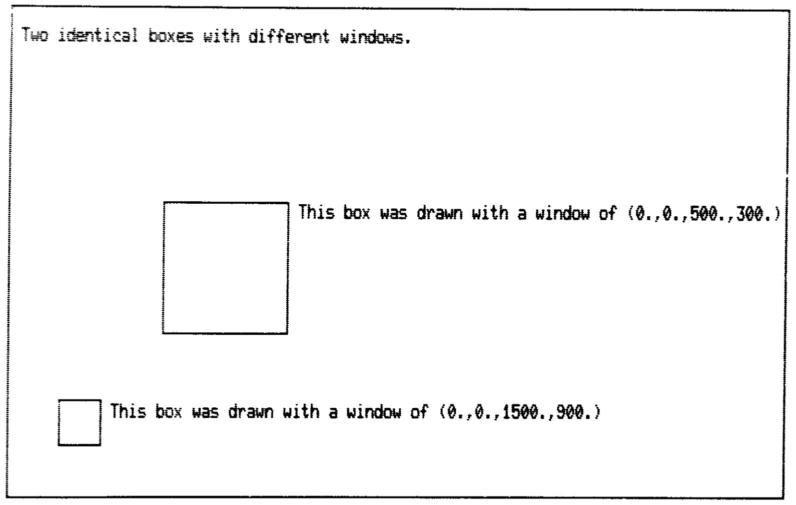
   means that xleft is equal to xright or that ybot is equal to ytop. The call is ignored and the previous coordinate system remains in effect.

PROGRAM EXAMPLE

The following program sets the window, draws a box, resets the window, and redraws the same box. See Figure 5–37.

```
C   Initialize VT125 and draw a box and a title on the screen.
          CALL INITGR (5)
          CALL CLRSCR
          CALL CLRTXT
          CALL BOX (0.,0.,767.,479.)
          CALL LINETX(2,2'Two identical boxes with different windows.')
C
C   Now call SWINDO and draw and label a box.
          CALL SWINDO (0.,0.,500.,300.)
          CALL BOX (100.,100.,180.,180.)
          CALL MOVE (180.,180.)
          CALL TEXT (' This box was drawn with a window of
          1 (0.,0.,500.,300.)')
C
C   CALL SWINDO for a different window but draw the same box.
          CALL SWINDO (0.,0.,1500.,900.)
          CALL BOX (100.,100.,180.,180.)
          CALL MOVE (180.,180.)
          CALL TEXT (' This box was drawn with a window of
          1 (0.,0.,1500.,900.)')
C
          END
```

Figure 5–37: Illustrations of SWINDO Calls


Two identical boxes with different windows.
This box was drawn with a window of (0.,0.,500.,300.)
This box was drawn with a window of (0.,0.,1500.,900.)

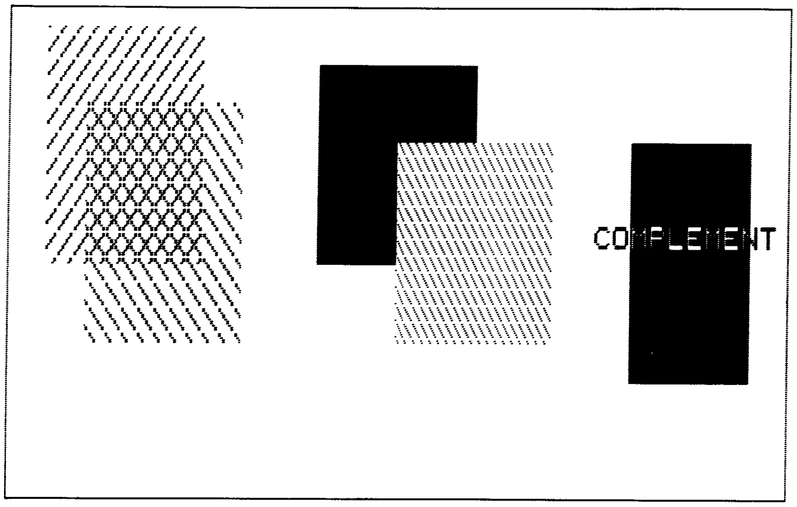
PURPOSE

SWMODE (*S*et *W*riting *Mode*) enables you to draw graphic objects in overlay, replace, or complement mode, to erase them, or to display them in reverse mode.

The overlay mode with the reverse writing turned off is the initial default. It is established as the default mode when you call the INITGR subroutine. (INITGR must be the first RGL/11 subroutine call in every program in order to set all the terminal's attributes to known values.) A writing mode remains in effect until changed by another SWMODE, or by an INITGR call.

Refer to Section 2.9, Using Writing Modes, for more information on how to use the SWMODE subroutine.

Figure 5–38: Examples of Overlay, Replace, and Complement Modes



FORM

SWMODE ('mode')

ARGUMENT DESCRIPTION

'mode'	is a two–character string expression that defines the writing mode:
'CO'	draws graphic objects in complement mode. All subsequent graphic objects are drawn in complement mode until you call SWMODE again and enable either the overlay, replace, or erase writing mode or until you call INITGR.
'ER'	draws graphic objects in erase mode. All subsequent graphic objects are drawn in erase mode until you call INITGR or SWMODE again and enable either the overlay, complement, or replace writing mode.

'IN'	enables you to reproduce the initial writing mode defaults in one step, not two; that is, it establishes overlay mode with reverse writing turned off as the current writing mode.
'NR'	cancels the reverse writing mode.
'OV'	draws graphic objects in overlay mode. All subsequent graphic objects are drawn in overlay mode until you call SWMODE again and enable either the complement, replace, or erase writing mode.
'RE'	draws graphic objects in replace mode. All subsequent graphic objects are drawn in replace mode until you call INITGR or SWMODE again and enable either the overlay, complement, or erase writing mode.
'RV'	writes graphic objects in reverse. What would normally be displayed as the screen color is displayed as the drawing color, and vice versa. Reverse mode can be used in combination with either the overlay, replace, complement, or erase writing modes (by issuing two consecutive calls to SWMODE). Reverse writing mode stays in effect until you call INITGR or SWMODE again and enable no–reverse mode.

EXAMPLE

```
SWMODE ('OV')
```

sets the writing mode to overlay mode.

RELATED SUBROUTINES

none

RESTRICTIONS

You can specify only one writing mode each time you call SWMODE. For example, if you want reverse and overlay writing modes to be in effect simultaneously, you must call SWMODE two times before drawing any graphic objects.

ERROR MESSAGES

1. Error Code 210

```
%RGL-W-INVMODE, 'x' is not a valid writing mode
```

means that you misspelled the writing mode or specified an undefined writing mode. The call is ignored and the current writing mode continues.

PROGRAM EXAMPLE

See Section 2.9, Using Writing Modes, for examples.

PURPOSE

TEXT displays in graphic text the message you specify, beginning at the current graphic cursor location. TEXT uses the current writing mode as the default writing mode, but it can use any writing mode you establish with SWMODE. To position the TEXT starting location where you want it, use the MOVE subroutine to move the graphic cursor to the top left pixel of the first character to be displayed.

After the message has been displayed, the graphic cursor is at the end of the message, at the top right of the last character.

See Section 2.8, Labeling the Picture, for more information.

FORM

TEXT ('string')

ARGUMENT DESCRIPTION

'string'	is a string expression of the message to be displayed. The length of the string must be 80 characters or less; any message that exceeds 80 characters is truncated at the 80th character. The string is written in the current writing mode. If you want to display a double quotation mark within the text string, you must pass two double quotation marks (" ").

EXAMPLE

```
TEXT ('Send this message')
```

displays the message

Send this message

starting at the current graphic cursor location.

```
TEXT ('Say " "Hello" " ')
```

displays the message

Say "Hello"

starting at the current graphic cursor location.

RELATED SUBROUTINES

SCHRST determines the character set TEXT uses.

STXSIZ determines the size of the characters.

CLRSCR erases TEXT's display.

LINETX displays a message with graphic characters, but their size is not affected by STXSIZ calls and LINETX begins the message at a location you specify.

RESTRICTIONS

The message is limited to 80 characters. Any message that exceeds 80 characters is truncated at the 80th character.

To include a double quotation mark as part of the message, you must precede it with another double quotation mark (" "). This is true whether you pass the message as a string literal or as a string variable.

ERROR MESSAGES

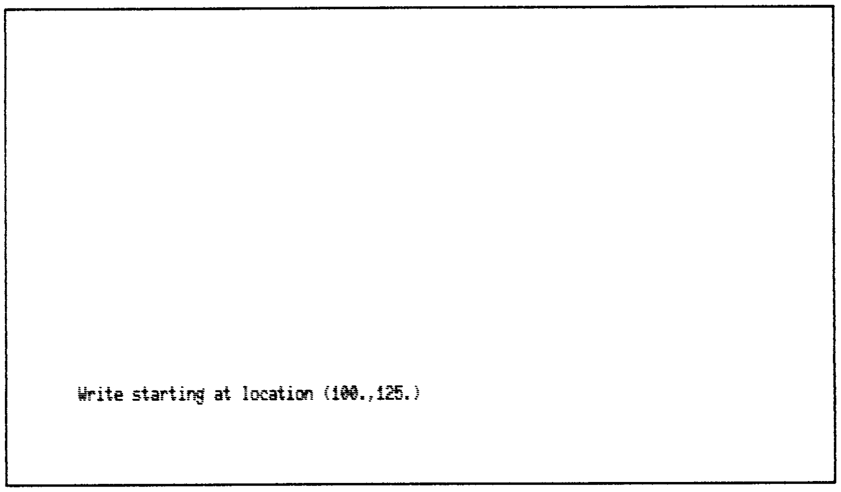
none

PROGRAM EXAMPLE

The following program fragment sets the window and writes a message starting at location (100., 125.). This location represents the upper left pixels of the first letter, "W". See Figure 5–39.

```
CALL SWINDO (0.0, 0.0, 1000.0, 625.0)
CALL MOVE (100.0, 125.0)
CALL TEXT ('Write starting at location (100., 125.)')
```

Figure 5–39: Illustration of a TEXT Call



```
Write starting at location (100.,125.)
```

Appendix A
Summary Of RGL/11 Subroutines

This appendix lists the RGL/11 subroutines in alphabetical order. The braces ({ }) indicate optional arguments; parentheses and commas are always required.

ARC (angle, x, y)
Draws an arc of the specified length beginning at the specified location. The center of the arc is at the current graphic cursor location.

ARCC (angle, x, y)
Draws an arc of the specified length beginning at the current graphic cursor location. The center of the arc is at the specified location.

BOX (x1, y1, x2, y2)
Draws a box that passes through the specified diagonally opposite corners.

CIRCC (radius, angle)
Draws a circle whose radius and angle of rotation are specified. The circumference passes through the current graphic cursor location.

CIRCLE (radius)
Draws a circle whose radius is specified. The center is at the current graphic cursor location.

CIRCXY (x, y)
Draws a circle whose circumference passes through the specified location. The center is at the current graphic cursor location.

CLRSCR
Clears all objects on the graphic (VT125) screen; regular text is not cleared.

CLRTXT

Clears all objects on the text–mode (VT100) screen; graphic objects are not cleared.

CPYSCR

Copies all graphic objects on the screen to the LA34–VA printer. (Regular text is not copied.)

DPAPER ({'xaxistype'}, {xa}, {xb}, {'yaxistype'}, {ya}, {yb}, {'gridcolor'})

Draws a grid on which to plot data.

GCLOSE

Closes the graphic file that was opened by a call to GSAVE.

GETLOC (x, y)

Returns the coordinates of the current location of the graphic cursor.

GETSTA (istat)

Returns the error code of the last RGL/11 error, if one has occurred; otherwise it returns "001, the code for an error–free condition.

GLOAD ('filespec', {LUN)}

Transmits to the terminal the commands that are stored in the specified file. The commands are in ReGIS format.

GSAVE ('filespec', {LUN})

Saves all subsequent graphic commands in the specified file. The commands are saved in ReGIS format.

INITGR (LUN)

Sets all graphic attributes to their default values. INITGR uses the LUN specified as the LUN for the input and output device. It usually uses 5, your terminal.

LCHRST (number, 'filespec', {LUN})

Loads a character set into the terminal from the specified file.

LFIXED (number, {xarray}, yarray, {'yaxis'}, indexarray)

Returns the coordinates of up to ten locations on a data path of a graph in the units of the graph's axes.

LFREE (x, y, key, {'yaxis'})

Returns the coordinates of a location anywhere on a graph in the units of the graph's axes.

LINE (x, y)

Draws a line from the current graphic cursor location to the specified location.

LINETX (irow, icol, 'string')

Displays a text string of standard size at the specified row and column.

LNAXIS ('axisid', {'axislabel'}, {minvalue}, {maxvalue}, {exact})
Assigns numeric values to the cells (divisions) along each axis and labels each cell boundary with that label. It also gives the specified axis a character label that identifies its function.

LOCATE (x, y, key)
Displays a cursor that can be moved with the arrow keys on the keyboard. Typing a non–arrow key returns the location of the cursor.

LTAXIS ('axisid', {'axislabel'}, minvalue, maxvalue, {maxchars}, {'string'})
Assigns numeric labels to the cells (divisions) of a graph's top or bottom x–axis and labels each cell boundary with a text label. It also gives the specified axis a character label.

MARKER ({number}, {x}, {y})
Displays a marker at the current graphic cursor location or at the specified location.

MOVE (x, y)
Moves the graphic cursor to the specified location.

PDATA (n, {xarray}, yarray, {'yaxis'}, {'colorname'}, {marker}, {linetype}, {smooth}, {shade}, {yvalue})
Maps the xarray and yarray data into x– and y–coordinates and plots them on a "sheet" of graph paper.

POLYGC (nsides, radius, angle)
Draws an n–sided regular polygon with one vertex passing through the current graphic cursor location.

POLYGN (nsides, radius)
Draws an n–sided regular polygon whose center is at the current graphic cursor location.

POLYGX (nsides, x, y)
Draws an n–sided regular polygon with one vertex at the specified location.

POLYLN (number, xarray, yarray)
Draws connected lines using X and Y arrays to define the endpoints.

PPOINT (x, y, {'yaxis'}, {marker})
Displays a marker at the specified x,y coordinates on the graph.

RELBOX (dx, dy)
Draws a box of the specified size.

RELLIN (dx, dy)
Draws a line from the current graphic cursor location to a location whose distance is dx,dy away.

RELMKR ({number}, dx, dy)
Displays a marker on the screen at a location whose distance is dx, dy from the current graphic cursor location.

RELMOV (dx, dy)
Moves the graphic cursor to a location whose distance is dx, dy away.

RELPLN (number, dxarray, dyarray)
Draws connected lines using dx and dy arrays to define the points which are to be connected.

SCHRST (number)
Selects a character set with which all subsequent graphic text is written; the number can be 0, the standard set, or 1, the Greek set.

SCOLOR ({'name'}, n)
Sets the color attributes for the terminal.

SDEBUG
Enables the displaying of error messages on the terminal.

SDGREE
Causes angles to be interpreted as degrees rather than radians.

SLNPAT ({number}, {mult})
Selects the line pattern for subsequent picture objects.

SNDBUG
Stops all error messages from being displayed on the terminal.

SNSHAD
Turns off shading.

SRADNS
Causes angles to be interpreted as radians rather than degrees.

SSHADE ({yline}, {ipat})
Selects shading and the shade pattern.

STXSIZ ({iwide}, {ihigh})
Sets the size of text written by TEXT.

SWINDO (xleft, ybot, xright, ytop)
Defines the portion of the coordinate system that is to be displayed.

SWMODE ('mode')
Selects the writing mode for drawing graphic objects.

TEXT ('string')
Displays the text string starting at the current graphic cursor location.

Appendix B
RGL/11 Error Messages

The initial RGL/11 error–handling procedure is to display the error message on the screen when an error occurs in a RGL/11 subroutine and also to set a status word indicating the status of the error. (The status word is returned to you only if you call the GETSTA (*Get Sta*tus) subroutine and read its argument.)

The error message is displayed at the bottom of the VT100 screen in regular text, not graphic text.

Once your program is debugged, you may not want error messages displayed on the screen. You may prefer to handle errors by calling the GETSTA subroutine. To stop error messages from being displayed on the screen, use the SNDBUG (*S*et *N*o *Debug*) subroutine.

The ISTAT argument of GETSTA always registers the code of the last error. It is set to “001, the status of an error–free condition, by the INITGR subroutine and by the GETSTA subroutine itself after it has returned the current error status.

NOTE

> All RGL/11 error messages are warnings; there are no fatal error messages. However, after an error message is displayed on the screen, you must type the RETURN key to have the program continue.
>
> If an error occurs when SNDBUG is in effect and the message is not displayed, the program does not halt; however, RGL/11 continues to set the error code in the ISTAT variable.

The RGL/11 subroutines cannot protect against bad data, such as a string that is not terminated by a null byte or integer coordinates used when floating point coordinates are required. These conditions cannot be detected by the subroutines and will usually cause unexpected results on the screen.

RGL/11 error messages begin "RGL–W–" to indicate that RGL/11 is the error–handler and that the error is classed as a warning (–W–), not a fatal error. The following list of RGL/11 errors does not include that identification. The 'x' in single quotes indicates a variable that the error message replaces with the appropriate value. The code numbers are in octal notation.

NOTE

If the program hangs in graphic mode and you use the VT125's SET–UP and reset keys to restart, your program may be adversely affected.

Error Code **Message and Circumstances that Caused the Error**

010 **CNF, color 'x' not found**
indicates that you misspelled the color name or referenced a nonexistent color name. The current drawing color is associated with the color number given and the program continues

Message can be generated by: DPAPER, PDATA, SCOLOR

100 **INVWINCOORD, invalid window coordinates**
indicates the following conditions in the call that caused the error:

LINE MOVE	You tried to move the cursor more than 32767 pixels. The call is ignored.
SWINDO	You specified either a leftmost x–value that was equal to the rightmost x–value or a bottom y–value that was equal to the top y–value. The call is ignored; the previous coordinate system remains in effect.

Message can be generated by: LINE, MOVE, SWINDO

110 **OUTRANGE, coordinates out of range**
indicates that the coordinate specifications for irow and icol were outside the number of screen lines and columns.

Message can be generated by: LINETX

150 **INVCHARSET, 'x' is not a valid character set**
indicates an attempt to select a character set that was greater than 1 or less than 0.

Message can be generated by: LCHRST, SCHRST

160 **INVSHADPAT, 'x' is not a valid shading pattern**
indicates that you specified a value for the shade pattern that was either (1) less than or equal to zero, (2) between "2 and "37 (octal), or (3) greater than "127 (octal).

Message can be generated by: SSHADE

Error Code	Message and Circumstances that Caused the Error
170	**INVTEXSIZ, 'x' is not a valid text size** indicates that either iwide or ihigh was less than 0 or greater than 16. If you gave ihigh an invalid value, RGL/11 sets it to 1.5 times iwide and continues. If you defaulted iwide or gave it an invalid value, RGL/11 sets iwide to 1 and ihigh to 1.5 times iwide and continues. Message can be generated by: STXSIZ
200	**INVLINE, 'x' is not a valid line pattern** indicates an attempt to specify a pattern multiplier that was outside the range of 2 through 9 or a line pattern outside the range of 0 through 9. Message can be generated by: PDATA, SLNPAT
210	**INVMODE, 'x' is not a valid writing mode** indicates that you either misspelled the writing mode or specified an undefined writing mode. Message can be generated by: SWMODE
220	**INVSIDES, invalid number of sides** indicates an attempt to specify nsides to be less than three. Message can be generated by: POLYGC, POLYGN, POLYGX
230	**OPENIN, error opening 'x' as input** indicates an attempt to specify a file that could not be located. Message can be generated by: GLOAD, LCHRST
240	**OPENOUT, error opening 'x' as output** indicates an attempt to specify an invalid filename. Message can be generated by: GSAVE
250	**RER, read error on unit 'x'** indicates an attempt to specify a file that could not be read. Message can be generated by: GLOAD, LCHRST
260	**WER, write error on unit 'x'** indicates that the file or LUN could not be written to. Message can be generated by: GSAVE, INITGR
270	**BADMARKER, 'x' is not a valid marker** means you specified a marker number that is less than 0 or greater than 10. RGL/11 uses the default marker, a dot, and continues. Message can be generated by: MARKER, RELMKR, PDATA, PPOINT

Error Code	Message and Circumstances that Caused the Error
330	**NOCHARSET, character set 'x' has not been loaded** indicates an attempt to enable a character set that had not been loaded. Message can be generated by: SCHRST
340	**INVNUMEL, 'x' is an invalid number of elements** indicates that you specified a value for the number–of–elements argument that was zero or negative. RGL/11 immediately returns control to the calling program. Message can be generated by: LFIXED, POLYLN, RELPLN, PDATA
460	**NCX, 'x' is an invalid number of cells (X axis)** indicates you specified an illegal number of cell divisions for the x–axis scale. The default is supplied and the program continues. Message can be generated by: DPAPER
470	**NCY, 'x' is an invalid number of cells (Y axis)** indicates you specified an illegal number of cell divisions for the y–axis scale. The default is supplied and the program continues. Message can be generated by: DPAPER
500	**NXT, 'x' is a nonexistent X axis type** indicates you specified an x–axis type that does not match any code defined for the 'xaxistype' argument. The default is supplied and the program continues. Message generated by: DPAPER
510	**NYT, 'x' is a nonexistent Y axis type** indicates you specified an y–axis type that does not match any code defined for the 'yaxistype' argument. Message generated by: DPAPER
520	**NSX, 'x' is an invalid number of subcells (X axis)** indicates you specified an illegal number of subcell divisions for the x–axis scale. The default is supplied and the program continues. Message can be generated by: DPAPER
530	**NSY, 'x' is an invalid number of subcells (Y axis)** indicates you specified an illegal number of subcell divisions for the y–axis scale. The default is supplied and the program continues. Message can be generated by: DPAPER

Error Code	Message and Circumstances that Caused the Error
560	**IMM, 'x' is an invalid maximum and/or minimum** indicates the following conditions in the call that caused the error: LNAXIS — You specified a value for minvalue that was greater than maxvalue. Autoscaling is invoked and the program continues. LTAXIS — You specified a value for minvalue that was greater than or equal to maxvalue. No action is taken and control returns to the calling program. Message can be generated by: LNAXIS, LTAXIS
570	**POW, all points are outside the window** indicates that all the data points passed to LFIXED lie outside the data–plotting window. When this occurs, no action is taken and control returns to the calling program. Message can be generated by: LFIXED
600	**AID, 'x' is an invalid axis identifier** indicates you specified an axis identifier code that does not match either the codes defined for the argument. If LFIXED, LFREE, PDATA, or PPOINT caused the error, the default is supplied and the program continues. If LNAXIS or LTAXIS caused the error, no action is taken and control returns to the calling program. Message can be generated by: LFIXED, LFREE, LNAXIS, LTAXIS, PDATA, PPOINT
610	**TCC, 'x' is too many characters per cell** indicates you included more characters in the 'string' argument than can fit between the first and last cell of a graph's axis. The maxchars argument defaults to the largest number of characters that are possible to write at each cell boundary without any of the labels overwriting each other. The first 64 letters in the string argument will be used. Message generated by: LTAXIS
640	**XNS, x–axis has not been scaled** indicates one of the following conditions prevailed: 1. If LFIXED, LFREE, or PPOINT caused the error, then the x–axis has not been scaled by LTAXIS, LNAXIS, or PDATA. 2. If PDATA caused the error, then: a. autoscaling was invoked but all values in the xarray are the same. b. the x–axis is logarithmic, autoscaling was invoked, but the xarray contained at least one non–positive number.

Error Code	Message and Circumstances that Caused the Error

When this error occurs, control returns immediately to the calling program.

Message generated by: LFIXED, LFREE, PDATA, PPOINT

650 **YNS, y–axis has not been scaled**
indicates the following conditions prevailed:

1. If LFIXED, LFREE, or PPOINT caused the error, then the y–axis has not been scaled by LNAXIS, LTAXIS, or PDATA.
2. If PDATA caused the error, then:
   a. autoscaling was invoked but all values in the yarray are the same.
   b. the y–axis is logarithmic, autoscaling was invoked, but the yarray contained at least one non–positive number.

When this error occurs, control returns immediately to the calling program.

Message generated by: LFIXED, LFREE, PDATA, PPOINT

740 **NPL, logarithm of non–positive number**
indicates the following conditions in the call that caused the error:

LNAXIS, LTAXIS	you specified a non–positive number for the minvalue or maxvalue argument when labeling a logarithmic axis.
PDATA	the input data array(s) contain a non–positive coordinate for a logarithmic axis.
PPOINT	one of the input variables was non–positive and was to be used for a logarithmic axis.

Message generated by: LNAXIS, LTAXIS, PDATA, PPOINT

750 **NOPAPER, no graph paper has been drawn**
indicates that you did not call DPAPER or PDATA before the subroutine that generated the message.

Message generated by: LFIXED, LFREE, LNAXIS, LTAXIS, PPOINT

770 **FIP, function is invalid with current graph paper**
indicates that the axis specified is already scaled. The call to LNAXIS or LTAXIS is ignored and control is returned to the calling program.

Message can be generated by: LNAXIS, LTAXIS

Glossary

Absolute location

a point on the screen whose x– and y–coordinates are based on its distance from the origin location; that is, its coordinates are measured as displacement from the (0.,0.) screen location, regardless of the location of the graphic cursor or text cursor. The coordinates' units are defined by the user. (See Origin location, Relative location.)

Alternate character set

the Greek character set.

Argument

a variable or constant in a subroutine call.

Argument default

the value for an argument used by a subroutine when that argument is not specified in the subroutine call. (See Initial default.)

Arrow keys

the keys in the upper right row of the main keypad that are usually used to move a cursor up, down, left, and right.

Autoscaling

the process the data plotting subroutines use to automatically select the range of numeric values against which to plot data.

Auxiliary keypad

the section of the keyboard to the right of the main keypad; it consists of a numeric pad and program function keys; the latter are often programmed for application–specific purposes. (See Program function keys.)

Brightness

the intensity of images on the terminal screen. You control screen brightness on the VT125 by pressing the up–arrow key to increase the brightness and pressing the down–arrow key to decrease it when the terminal is in SET–UP A or B mode.

Byte expression

a data type indexed on single byte boundaries, that is, a data type that requires only eight bits of storage.

Cell

a major division of a graph's axis. A cell boundary is displayed as a long tickmark or a solid line across a graph. (See Subcell, and the LNAXIS or LTAXIS subroutine.)

Character

an 8–bit ASCII code. For this terminal, displayable characters are represented by an eight–by–ten dot pattern that is the basic unit in each of the four character sets.

Character cell

a block of eight pixels by ten.

Character cell origin

the top left corner of a character cell.

Character set

the set of standard English alphanumeric characters and special characters such as commas, periods, percent signs; or the alternate–character set of Greek letters.

Character size

the width and the height of a character. (See the STXSIZ subroutine.)

Clipping

a calculation used by the data–plotting subroutines to cut off (make invisible) the data line of a graph at the data plotting window, and by the picture drawing subroutines to clip lines at the screen boundaries.

Complement writing

one of the writing modes that allows two intersecting graphic objects to be highlighted. (See the SWMODE subroutine.)

Coordinate

one of two values that define a location. RGL/11 uses two coordinate schemes: world coordinates and screen coordinates.

Coordinate pair

an x–coordinate and a y–coordinate that together define a location.

Current character set

the most recently specified character set. (See SCHRST subroutine.)

Cursor keys

see Arrow Keys.

Cycle (logarithmic)

the numbers in a single power of ten.

Default

a value the system supplies to certain variables (parameters, attributes) when no choice is specified for that variable in the subroutine call or when the choice was invalid. The defaults that the system supplies when it is first loaded are the "initial defaults." Defaults that use values from a previous subroutine call are called "argument defaults."

Drawing color

a shade of gray used to draw graphic objects. If a color monitor is attached to the VT125, the gray shades map to colors.

Erase mode

one of the writing modes; it removes previously drawn objects.

Expression

an argument in a command statement of a RGL/11 subroutine; it can be either a constant or a variable.

Floating–point expression

an expression that requires real (decimal) numbers; the decimal point is a required part of the number.

Graphic cursor

a blinking, diamond–shaped cursor that indicates that the terminal is operating in graphic mode.

Graphic cursor location

the location RGL/11 maintains internally that is a pointer to the location on the screen last moved to or displayed.

Graphic mode

one of two modes of the VT125 terminal which RGL/11 uses. In this mode, the ReGIS interpreter is enabled to perform graphics.

Graphic object

any portion of a picture that you perceive as an entity. Examples are a line, circle, or a message written in graphic text; there are the two types of graphic objects: picture objects and graphic text.

Graphic screen

or VT125 screen, the screen to which the terminal writes when it is in graphic mode. (See VT100 screen.)

Graphic text

text written and displayed when the terminal is in graphic mode; It differs from regular text (text you write when the terminal is in text mode) in that you have more ways to display it, it is displayed on the graphic screen, and it is erasable by the CLRSCR call rather than by CLRTXT.

Gray scale

the levels of intensity the screen can display with the standard black–and–white monitor.

Image

all objects that are displayed on the screen, that is, any combination of drawings and text displayed on the screen.

Initial default

the default value established by the INITGR subroutine. (See Argument default.)

Keyboard

the main keypad and auxiliary keypad of the VT125.

Line pattern

the sequence of dots and dashes used in drawing a line. (See SLNPAT, the subroutine that sets the line pattern.)

Location

a point defined by the x and y coordinate pair.

Locator cursor

the cursor displayed by the LOCATE, LFIXED, and LFREE subroutines.

LUN

Logical Unit Number. (See the Preface, Documentation Conventions, for LUN numbers.)

Main keypad

that portion of the VT125 keyboard above the space bar and separated from the auxiliary keypad; it consists of the alphanumeric and special–character keys.

Mnemonic

an abbreviation or acronym that is easy to remember.

Monitor

a video device containing a Cathode Ray Tube (CRT) which the terminal uses to display screen images.

Offset

the distance from a given location; it is usually used to define locations relative to the current graphic cursor location.

Optional arguments

arguments that do not have to be specified in a RGL/11 subroutine call; however the punctuation that delimits them must be specified.

Origin location

the location of the first x–coordinate and first y–coordinate. The default origin location is at the bottom left.

Overlay writing

one of the writing modes; it provides a user with a transparent overlay of two intersecting graphic objects. (See the SWMODE subroutine.)

Physical screen coordinates

the coordinates the terminal uses to define how the screen is addressed.

Picture

any combination of drawing and text that is displayed on the screen.

Picture object

any graphic object except graphic text, for example, lines, arcs, polygons, circles.

Pixel

picture element; it is the smallest displayable unit on the VT125 screen.

Program function keys

top four keys on the auxiliary keypad, labeled PF1, PF2, PF3, and PF4. PF3 and PF4 are used by the LFREE, LFIXED, and LOCATE subroutines.

Radian

a unit of angular measure. One radian is equal to 360 degrees divided by two pi, or approximately 57 degrees 17 minutes. A circle is an arc of two pi radians. (See SRADNS subroutine.)

ReGIS

the *R*emote *G*raphics *I*nstruction *S*et, the set of internal commands used by the RGL/11 subroutines to draw pictures and plot data.

RGL/11

the *R*eGIS *G*raphics *L*ibrary for the *F*ORTRAN *E*nhancement *P*ackage, Version 2.

Relative location

a point on the screen whose coordinates are based on its distance from the current graphic cursor location rather than from the origin location. The coordinates' units are defined by the user. (See Absolute location.)

Replace writing

one of the writing modes; it forces the pattern of a graphic object onto the terminal screen no matter what is already on the screen. (See the SWMODE subroutine.)

Reverse writing

one of the writing modes; it reverses the intensity of the colors.

Screen

that portion of the video monitor on which images are displayed. The screen can display two kinds of images simultaneously: (1) text images created when the terminal is acting like a text–processing ASCII terminal (and the screen is a "VT100 screen"), and (2) images displayed when the terminal is operating as a graphic terminal (and the screen is a "VT125 screen").

Screen color

the background shade of gray of the screen; it is usually black, GRAY0, the absence of any light.

Shade character

a user–selected character that the terminal uses when it shades picture objects. (See the SSHADE, SLNPAT, and PDATA subroutines.)

Shade line

the y–coordinate that delimits the area to be shaded during a write operation. (See the SSHADE subroutine.)

Shade pattern

the line pattern or character the terminal uses during a write operation when the shading option is enabled. (See the SLNPAT and SSHADE subroutines.)

Standard character set

the 128 ASCII character set: the alphanumeric characters, punctuation characters, and special characters; of the 128, 95 are displayable.

Subcell

a division of a cell; that is, a subdivision in a graph's axis. A subcell boundary is displayed as a short tickmark or a dotted line across a graph. (See Cell and the LNAXIS subroutine.)

Subtend

to be opposite to and to delimit. For example, the side of a triangle subtends the opposite angle.

Text cursor

a VT100 cursor that indicates the terminal is in text mode and also indicates where on the screen the text will be displayed.

Text mode

one of the two operating modes of a VT125 terminal that RGL/11 uses; in text mode the terminal operates as a standard ASCII text terminal.

Vector

a directed line, specified by a magnitude and a direction. The numbers (x, y) are components of the vector.

VT100 screen

the screen addressed when the VT125 terminal is in text mode.

VT125 screen

the screen addressed when the VT125 terminal is in graphic mode.

Window

the part of the user's picture that RGL/11 displays. When RGL/11 is entered, the window is set to be the same as the screen's coordinates; the window is the full screen. (See the SWINDO subroutine.)

World coordinates

the user–specified coordinate range that defines how the screen is addressed.

Index

RGL/11 Programmer's
Reference Manual
AA–M837A–TC

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 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 ____________ Telephone ____________

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

SOFTWARE PUBLICATIONS
200 FOREST STREET MR01-2/L12
MARLBOROUGH, MA 01752

----- Do Not Tear – Fold Here and Tape --