Burroughs

Programmer's Guide

B 20 Systems Graphics

(Relative to Release Level 4.0)

Distribution Code SA

Priced Item
Printed in U.S.A.
May 1985

1180098

Burroughs

Programmer's Guide

B 20 Systems Graphics

(Relative to Release Level 4.0)

This manual replaces all previous editions of 1180098.

Distribution Code SA

Priced Item
Printed in U.S.A.
May 1985

1180098

The Graphics Support Package contains software routines that drive the following hardware peripherals supported by Burroughs Corporation:

Burroughs AP1351 Multi Function Printer

Burroughs B9253 dot matrix printer

The Graphics Support Package also contains software routines which drive the following hardware peripherals:

Hewlett-Packard Model HP7220C 8 pen plotter

Hewlett-Packard Model HP7220T 8 pen plotter

Hewlett-Packard Model HP7470A 2 pen plotter

Hewlett-Packard Model HP7475A 6 pen plotter

Strobe Model 100 1 pen plotter

Printronix MVP dot matrix printer

Envision 420 dot matrix printer

Anadex 9620 dot matrix printer

Okidata Microline 93 dot matrix printer

Dataproducts 8010 dot matrix printer

The particular device selected is the responsibility of the customer.

Correspondence regarding this publication should be forwarded, using the Documentation Evaluation Form at the back of the manual, or remarks may be addressed directly to Burroughs Corporation, Corporate Product Information East, 209 W. Lancaster Ave., Paoli, PA 19301, U.S.A.

LIST OF EFFECTIVE PAGES

Page	Issue
iii	Original
iv	Blank
v thru ix	Original
x	Blank
1-1 thru1-3	Original
1-4	Blank
2-1 thru 2-12	Original
3-1 thru 3-63	Original
3-64	Blank
4-1 thru 4-18	Original
5-1 thru 5-15	Original
5-16	Blank
A-1 thru A-3	Original
A-4	Blank
B-1 thru B-6	Original
C-1 thru C-5	Original
C-6	Blank
D-1	Original
D-2	Blank
E-1, E-2	Original
F-1	Original
F-2	Blank
1 thru 4	Original

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1
OVERVIEW

Burroughs graphics software products support a wide range of graphics functions. Using a modular architecture that distributes the processing between the host CPU and the graphics control board, the graphics software provides two levels of support:

- Business Graphics, a high-level, menu-driven graphics application system that can be accessed from user-designed applications.

- the graphics library, a set of system-level procedures that can be called from user-designed applications to use all the capabilities of the graphics software.

This guide focuses on the second level of graphics support, the graphics library (Graphics.Lib). Using the procedures in this library, system designers can access the full range of graphics functions to develop high-performance, flexible application systems. The graphics library procedures are used to draw vectors and arcs and, on the B 22, to manipulate rectangles of bits. A variety of colors, line types, drawing modes, and fill patterns are available. Text labels and annotations can be written with variable attributes for font, color, character size, and label origin. Once a graphic representation has been created, it can easily be scaled and translated to assume different sizes, shapes, and positions on the display. These transformations are handled independently and do not alter the original definition of the figure.

Graphic representations can also be viewed in a variety of ways. Small sections can be magnified, and the size and shape of the display can be changed dynamically. Complex graphic representations can be built by merging different figures in the same display. Graphic representations created with graphics library procedures can also be filed for future use. In addition, they can be plotted on publication-quality pages or on transparencies and printed on dot matrix printers.

Business Graphics is a Burroughs application system that uses graphics library procedures to perform graphics functions for business applications. Business Graphics must be used in conjunction with an application system that extracts statistical data or figures from a data base and arranges this data in a tabular format. Business Graphics is invoked to convert the tabular data into graphic representations such as line charts, bar charts, and pie charts. Business Graphics can be accessed through Multiplan or another Burroughs software product, or system developers can create their own applications to interface with Business Graphics. Detailed information about the features of Business Graphics is found in the Business Graphics Reference Manual. In addition, Section 5 of this guide explains how application systems can be modified to pass data to Business Graphics for the generation of graphic representations.

B20GS4

Highlights

B20GS4 is the package for the Burroughs 4.0 Graphics Support Package. It can be installed on the XE520 as well as B 20 and B 26 workstations.* The package is created on one floppy disk for both the 8 inch and 5-1/4 inch medias.

Installation Procedures

To install B20GS4 on the XE520, do the following:

1. Insure that all cluster workstations are powered off.

2. Use either the XE520 BTOS User's Guide (form 1166295) or the XE520 System Administrator's Handbook (form 1166311) to determine installation procedures. These procedures involve booting a clustered workstation.

3. Follow steps 1 and 2 on the next page.

* Note that while the software is installed on the XE520, it actually runs on the attached B 20 series workstation. Therefore, when this manual refers to B 20 series workstations, such references include B 20 workstations that are attached to the XE520.

To install B20GS4 on any workstation, other than a B 26 Dual Floppy Standalone, the procedure is the same for both the 8 inch and 5-1/4 inch medias:

1. Insert the B20GS4 floppy disk into the appropriate disk drive, F0;
2. type in the command Software Installation and press GO.

At this point, you receive these prompts:

```
INSTALLATION OF BURROUGHS GRAPHICS PACKAGE

PRESS GO WHEN READY
```

After the apropriate response, FdSys.Version is appended to Sys.Version. All the files in directory [F0]<Burroughs> are then copied to directory <SYS> on the system disk.

Next you see the following message:

```
INSTALLATION OF BURROUGHS GRAPHICS PACKAGE IS NOW COMPLETE
```

You can then use the newly installed software to write programs that invoke graphics calls.

File Contents

The file contents for B20GS4 are the same for the floppy disk in both the 8 inch and 5-1/4 inch medias, with two exceptions as noted. These files are listed as follows:

```
<Sys>fileHeaders.sys
<Sys>mfd.sys
<Sys>log.sys
<Sys>sysImage.sys
<Sys>bootExt.sys  **
<Sys>badBlk.sys
<Sys>crashDump.sys
<Sys>DiagTest.Sys  **
<Sys>FdSys.Version
<Sys>Grfx-5.0-Update.Sub
<Sys>Install.Sub

<Burroughs>Graphics.Fonts
<Burroughs>Graphics.Lib
<Burroughs>ComplexRoman.font
<Burroughs>DuplexRoman.font
<Burroughs>Gothic.font
<Burroughs>SimplexPlot.font
<Burroughs>SimplexRoman.font
<Burroughs>HPPlotterConfig.sys
<Burroughs>PlotterConfig.sys
<Burroughs>StrobeConfig.sys
<Burroughs>StrobePlotterConfig.sys
<Burroughs>GraphicsPrinterConfig.sys
```

** - These files are found only on 8 inch floppy disks and are not on 5-1/4 inch floppy disks.

SECTION 2
CONCEPTS

GRAPHICS LIBRARY

The graphics library contains two different types of procedures. The main portion of the library is a set of device-independent procedures. Programs calling these procedures can be executed on any B 20 Graphics workstation. Many of the features described in the "Overview" section, such as saving graphic representations in files, translating and scaling figures on the screen, and viewing graphic representations from different perspectives, are supported only by the device-independent procedures. Detailed descriptions of the functions supported appear in the "Device-Independent Procedures" subsection that follows. The actual procedural interfaces for these commands are included in Section 3, "Device-Independent Procedures."

The graphics library also contains device-independent procedures that are called by the graphics software. These called procedures can be replaced by user-written routines to expand the capabilities of the software. They are used primarily to provide the end user with messages or instructions about operating the output devices. These user-written procedures are also included in Section 3, "Device-Independent Procedures."

The other procedures found in this library are device-dependent. The use of these procedures is restricted to the following workstations:

* B 21 (color)
* B 22 (monochrome)
* B 26 (color and monochrome)

Detailed descriptions of these low-level procedures appear in the "Device-Dependent Procedures" (the following subsection). The actual procedural interfaces for these commands are included in Section 4, "Device-Dependent Procedures."

The device-dependent procedures are executed on the graphics control board. These procedures include bit manipulation functions that are mapped directly to the video display screen and are, therefore, only available at this level. The device-dependent procedures execute faster than the device-independent procedures. They are particularly useful for applications that use animation or require custom-designed fonts. Using device-dependent commands does, however, preclude the use of some of the features of the device-independent commands. Graphic representations do not include text labels, and they cannot be transformed, saved, or viewed from different perspectives.

Applications designed for the B 22 Graphics workstation can use device-dependent procedures that support user-defined fonts. These bit-mapped fonts can be used to create character representations not available in the standard 10-by-15 pixel alphanumeric font.

Both the B 21 graphics control board and the B 26 graphics controller module contain color mappers. Color mappers enable multicolor graphic representations to be displayed on the video screen. The device-dependent color procedures allow the use of up to eight colors at a time.

In addition, there are device-dependent procedures that use the color style RAM on the B 21 graphics control board as well as the B 26 graphics controller module. These procedures combine color selection with display attributes for alphanumeric data. Eight different combinations of color and attributes can be used on the screen at one time.

Device-Independent Procedures

There are two main concepts that are important in understanding the use of the device-independent procedures. First, there are the structural components of the graphic representations. These components are called pictures and objects. The other key concept is the use of device-independent coordinate systems rather than the coordinates of the physical device on which the graphic images are displayed. These two concepts are discussed below.

Pictures and Objects

In device-independent procedures graphic representations are called pictures. A picture is composed of one or more objects. One bar chart on the screen, for example, is a picture with one object. A pie chart, a line chart, and a bar chart all together on the screen is a picture with three objects. Objects can also overlay each other in pictures. If the graphic representation is to be saved, a picture must be opened before any drawing or text labeling can be performed. Once a picture is open, objects can be created.

An object is a set of graphics commands and labels that can be edited and manipulated as an entity. Although several objects can be present in the same picture, only one object can be created or edited at a time. As an object is constructed, information about its structure is accumulated. Each object has the following components:

o a list of vector and text commands

o a list of labels (text and attributes)

o a list of transformation values

Vector List. The graphic's portion of the data representation is collected here. The vector list includes commands such as Move, Draw, FillRectangle, and SetColor, which are used to create a graphic representation. Drawing attributes such as line type, drawing mode, and color are also saved. In addition, text that is not to be modified is put in the vector list. Individual commands within the vector list cannot be modified, but the entire list can be cleared and rebuilt to modify the object.

Label List. Labels are textual explanatory notes that accompany the vector portion of the object. The label list consists of the text and the attributes for each label. The attributes are characteristics of the label such as font name, character size, and label origin. Individual labels within the label list for an object can be added, deleted, or modified.

The alphanumeric labels and the vector list for an object are mapped to display memory, logically ORed, and displayed together. Because the text and vector commands are stored in different lists, these two components of an object are processed independently of each other. There are several different types of modifications that can be made to label text. Changing the font, changing the actual text, and moving the label, for example, can all be accomplished without altering the vector portion of an object.

Transformation List. Once an object has been created, transformation procedures can be used to alter the object's size, shape, and position within the picture. The transformation procedures translate and scale the object and save the translation and scalar units in a transformation list. The original specifications for the object, which are kept in the vector and label lists, are unchanged.

Drawing Attributes

Attributes are variable characteristics of an object. Drawing attributes are used with drawing procedures to provide more variety and contrast in graphic representations. These attributes are saved with drawing commands in the vector list. The attributes that are used with drawing procedures are line type, drawing mode, and color.

Line Type. The graphics software includes eight line types. A solid line is the default, and there are other patterns of dots and dashes. Figure 2-1 illustrates the eight line types provided with the graphics software.

Drawing Mode. The drawing mode describes the method by which a vector or arc is written to display memory. When the bits that form the vector have been calculated according to the line type, they are compared to the existing memory bits and written to display memory according to the drawing mode. There are four drawing modes: set, clear, complement, and replace. Figure 2-2 shows examples of the same pattern written to the display memory in each of the four modes.

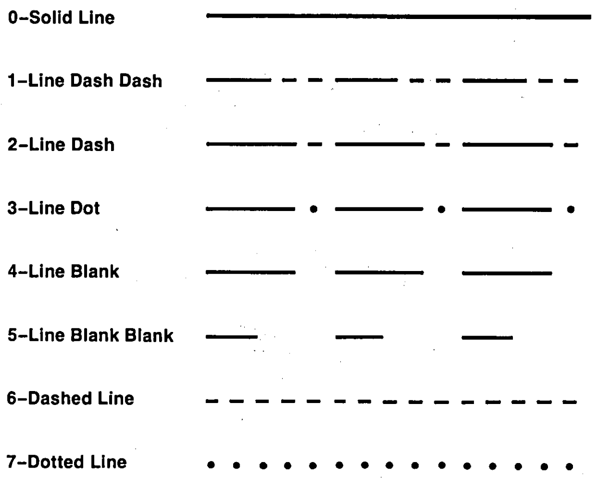


Figure 2-1. Line Types

Set mode logically ORs the pattern of the line to be drawn with the backgound bits already in the memory location. Thus, the bits that are "off" in the line pattern have no effect on the bits already in memory. Any bits that are "on" in the line pattern or the background remain on when they are merged.

Clear mode causes the bits that are on in the line pattern to turn the corresponding memory bits off.

Complement mode causes the line pattern to take on the opposite characteristic of the corresponding memory bits. The line pattern is logically XORed with the background display memory.

Replace mode is different from the other three modes in that no logical operation is performed between the line pattern and the existing memory. Whatever is in the line pattern replaces the existing memory bits. The background has no bearing on the line pattern.

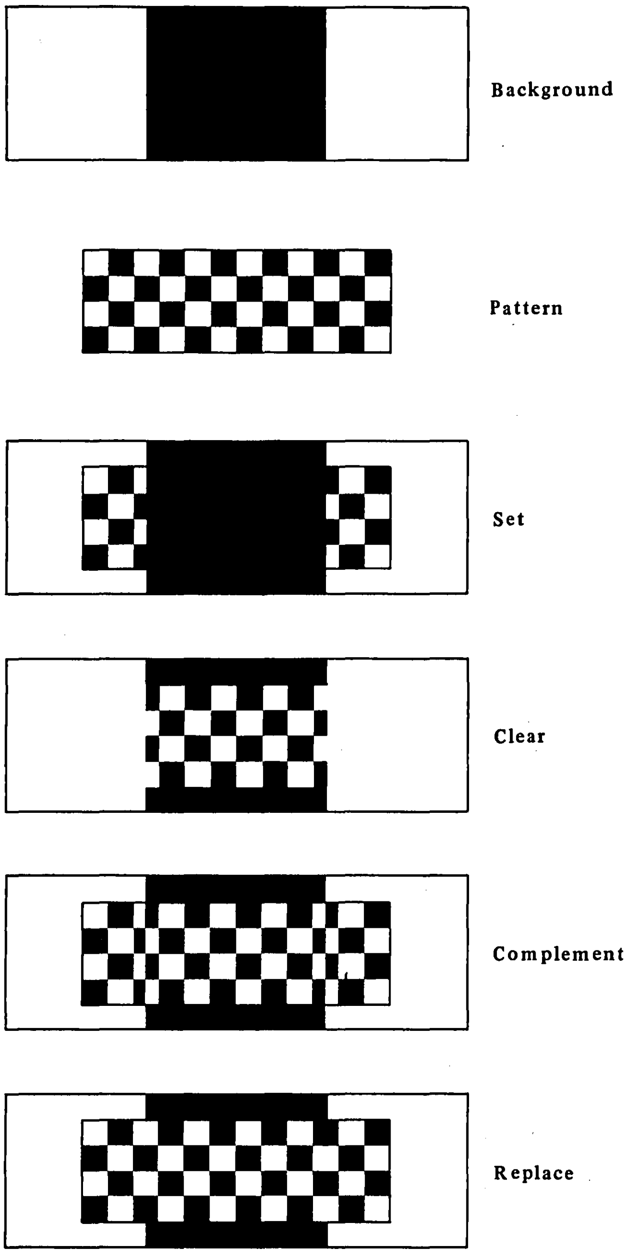


Figure 2-2. Drawing Modes

Color. The use of multiple colors is supported in both video display and plotter output on the following Color Graphics workstations:

* B 21 (color)
* B 26 (color)

On these display screens, 64 colors are available, any eight of which can be displayed at a time. The set of eight colors is called the color palette. Color selection is supported in the device-independent procedures by selecting a color from the current palette. Refer to the "Device-Dependent Procedures" subsection following for information specifying color palettes. If the output device is a plotter, the color parameters are used to specify pen numbers. The user can select a maximum of eight colors and assign a pen number for each one.

Text Attributes

Attributes are also used with text strings and labels to provide variety and contrast. The attributes associated with text strings are saved in the vector list for an object, and the label attributes are saved in the label list. The attributes that are used with text strings and labels are character size, font, and label origin.

Character Size. The standard alphanumeric font uses a character cell with a height of 36 pixels. The default character size is 1 for this standard size, but characters can be enlarged or reduced proportionally by specifying other values. Character size 2, for example, produces characters that are twice as high as the standard size, and .5 characters are half the size. The character size attribute functions as a scaling factor when an object is transformed. The characters maintain the same proportions in relation to each other and to their cells when an object is scaled to a smaller size.

Font. The graphics software includes four fonts: SimplexRoman, ComplexRoman, DuplexRoman, and Gothic. These font names are the internal names. The internal name is the name that is used by the graphics software and saved in pictures. User-friendly names such as Standard, Complex, Bold, and Gothic can also be defined for these fonts. With the exception of the standard default font, SimplexRoman, the file specifications for each font can be modified.

Label Origin. The label origin attribute indicates how text should be oriented in relation to the current display position. Text can be placed left flush, right flush, or centered at the current position, and it can begin at the top, middle, or bottom of the current position. Figure 2-3 illustrates the label origin positions.

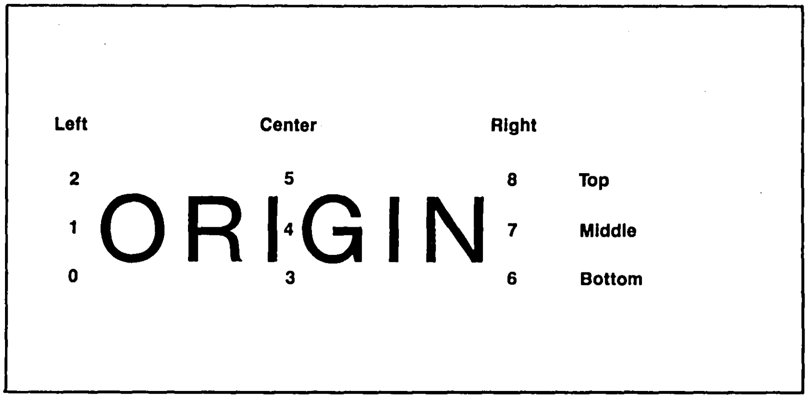


Figure 2-3. Label Origin

Picture File

Multiple objects can be transformed and merged on the display screen to create complex pictures. A completed picture, whether simple or complex, is saved in a picture file. Using the picture file eliminates the need to call all of the procedures used to create an object each time the picture is needed. The picture file can be opened repeatedly to change the way the picture is viewed and to modify or transform the objects within it.

Temporary Objects

Objects can also be defined as temporary. Temporary objects are used only when a picture is not open. For quick graphic representations used in testing, demos, or initial system development, this definition provides more efficient processing. The commands are performed to display the object, but no information is accumulated in vector or label lists to be saved in a picture file. Temporary objects also cannot be transformed, and only the standard font is available.

Device-Independent Coordinate Systems

To insure that applications that are written using device-independent procedures can run on any B 20 Graphics workstation, device-independent coordinates are used for mapping vector and text positions. The three different coordinate systems used to support output to workstation display screens and other devices such as plotters are:

- world coordinate system
- user-defined coordinate systems
- normalized device coordinate system

World Coordinate System. The world coordinate system is the primary system used internally by device-independent graphics library procedures. When an object is created, modified, or transformed, its position in the world coordinate system is mapped to display memory and saved in the vector list, label list, or transformation list. The world coordinate system theoretically maps objects to a 100-by-100 area. Position (0,0) is the lower left corner of the area, and the upper right corner is position (100,100). Coordinate units are specified as real numbers within this range.

Because the video display screens are not square, only the portion of the world coordinate system that represents the aspect ratio, or the ratio of height to width for the screen, is generally used. The coordinate positions that represent the aspect ratio of the B 22 screen are (0,0) to (100,77.74). The range for the B 21 series screen is (0,0) to (100,73.84). The range for the B 26 series screen is (0,0) to (100,74). Because the B 22, B 21, and B 26 series Graphics workstations have different aspect ratios, applications that are designed to run on both types of workstations must use the smaller range of (0,0) to (100,73.84) that is required for the B 21 series Color Graphics workstation.

User-Defined Coordinate Systems. The device-independent graphics library procedures also support user-defined coordinate systems. Once the user defines the minimum and maximum X and Y coordinate units, the parameters in subsequent procedures used to draw objects are interpreted as user-defined coordinates. The graphics software automatically converts the user-defined coordinates to the corresponding world coordinates.

Normalized Device Coordinate System. This system is used to reference the display area in a relative way. The coordinate positions range from (0,0) at the lower left corner to (1,1) at the top right. Once again, to maintain the aspect ratio of the screen, positions (0,0) to (1,.7774) for the B 22, (0,0) to (1,.7384) for the B 21 series, and (0,0) to (1,.74) for the B 26 series are the default ranges for the graphics workstations. The coordinate units actually describe positions in terms of their relation to the top, bottom and left sides of the display area. They are used to reference the video display screen, not a picture that is to be displayed. Currently, the only procedures that use this coordinate system are the cursor control functions and the viewport procedures. The coordinates used to place the cursor in the middle of the B 22 video display area, for example, are (0.5,0.39).

Viewing Perspectives

Pictures can be viewed dynamically from a wide range of perspectives. These viewing capabilities are invoked by adjusting the size of the world coordinate system window and the screen display area viewport. The window is a portion of the world coordinate area It defines what is to be displayed. The viewport is a portion of the screen. It defines where the information in the window is to be viewed. The information is displayed by scalingthe world coordinate values within the window to fill the viewport. The window/viewport transformations enable pictures to be viewed from many different perspectives. These viewing functions do not affect the picture data. The vector, label, and transformation lists for the objects within the picture are not altered.

The maximum size for the window is ordinarily the portion of the world coordinate area that corresponds to the aspect ratio of the display screen. Coordinate positions (0,0) to (100,77.74) for the B 22 Graphics workstation, positions (0,0) to (100,73.84) for the B 21 series Color Graphics workstation, and positions (0,0) to (100,74) for the B 26 series Graphics workstation are the ranges that represent the aspect ratios of the two different screens. The window can be set and reset to define different portions of the entire world coordinate area. For a window that defines the lower left quadrant of the B 22 display area, for example, the the boundaries are (0,0) to (50,38.87). No matter what objects have been mapped to positions in the world coordinate area, only the coordinates surrounded by the window are viewed. All of the coordinate positions outside the window are clipped.

The maximum viewport is the entire screen area. However, the viewport can be set and reset to define any rectangular portion of the screen where a picture is to be displayed. Ordinarily, part of the screen area is reserved for messages and forms. Therefore, the viewport is usually less than the whole screen area.

The perspective for viewing a picture can be altered dynamically as often as needed by adjusting the window and viewport sizes, shapes, or positions. A large picture can be scanned, for example, by keeping the size of the window constant but changing its position within the world coordinate area. A small section of a picture can be magnified by keeping the viewport large and resetting the window to surround only the portion of the picture that is to be enlarged. When the window and viewport are the same shape, the picture is viewed as it appears conceptually in the world coordinate area. When the window and viewport have dissimilar aspect ratios, the viewed picture is an oblique version of the original.

Device-Dependent Procedures

Device-dependent procedures support high-speed graphics functions. Device-dependent procedures can be used to draw vectors and arcs. These functions are device-dependent because they are executed entirely by the graphics control board firmware. Graphic representations are drawn using coordinates on display screens within the following workstations:

* B 21 (color)
* B 22 (monochrome)
* B 26 (color and monochrome)

Device-dependent procedures should, therefore, be used only when the code does not need to be transportable. The advantages of using these procedures are that they execute faster and provide color selection that is not available with the device-independent procedures.

The device-dependent graphics library procedures fall into four main functional categories: control functions, vector and arc manipulation functions, color functions, and alphanumeric attributes functions. The color procedures and alphanumeric attributes procedures can be used only on the following Color Graphics workstations:

* B 21 (color)
* B 26 (color)

Control Procedures

These procedures are used to control the output to the video display screen. For the B 22 Graphics workstation, there are procedures to control the two 64k bit-mapped planes. Only one plane can be displayed at a time. The displayed plane is called the visible plane. Either the visible plane or the invisible one can be defined as the current plane. The current plane is the destination for the operations creating a graphic representation.

Vector and Arc Manipulation Procedures

Vectors and arcs are plotted by calculating lines between endpoints. On the B 22 video display the current display plane is mapped using coordinate positions for a screen resolution of 656-by-510 pixels. Coordinate position (0,0) is the lower left corner of the screen, and coordinate position (655,509) is the top right corner. The B 21-series display memory has a screen resolution of 432-by-319 pixels. Coordinate position (0,0) is the lower left corner and (431,318) is the top right.

The B 26 series display memory has a screen resolution of 718-by-348 pixels. However, coordinate position (4,0) is the lower left corner (1435,1043) and is the top right. There are two coordinate positions per pixel horizontally and three coordinate positions per pixel vertically on a B 26.

Different line types, drawing modes, and colors can be set.

Refer to the "Drawing Attributes" subsection above for detailed information about the line type, drawing mode, and color options.

Color Procedures

The B 21 and the B 26 Color Graphics workstations support the use of 64 different colors, any eight of which can be displayed simultaneously in a picture. The color procedures allow for the definition of 8-byte color palettes that have one byte for each of eight colors. The bit settings in each color byte are interpreted by the color mapper and then displayed by the color monitor (on either the B 21 graphics control board or the B 26 graphics controller module).

Two 8-color palettes can be defined at a time, and individual colors in a palette can be replaced or modified. Color palettes can be accessed by device-independent procedures such as SetColor, but the specification of the colors that make up a palette is handled only by low-level functions. The color palette is saved with a picture in the picture file enabling pictures to be redisplayed with the same color specifications.

Alphanumeric Attribute Procedures

These procedures can be used only on the B 21 or the B 26 Color Graphics workstations. Text can be displayed in color independently or in conjunction with graphic representations. The alphanumeric attribute procedures allow the standard alphanumeric style RAM to be overridden by a color style RAM (on either the B 21 graphics control board or the B 26 graphics controller module). Eight different colors can be defined at one time along with different combinations of the reverse video and underlining attributes.

OUTPUT DEVICES

In addition to the workstation display screen, the graphics software also enables output to plotters and dot matrix printers. The plotters that can be interfaced with graphics for output are the Hewlett-Packard models 7470A, 7475A, 7220C, and 7220T, and the Strobe 100. Output can be plotted on paper or transparencies, and it can be written to a disk file. User-written procedures can be combined with device-independent graphics library procedures to provide messages and instructions on the screen for end users to load paper or change pens. Refer to Appendix B, "Using a Plotter" for detailed information on connecting and operating a plotter.

The only dot matrix printers that are supported for use with graphics are the Burroughs models AP1351 and B9253.

The dot matrix printers that can be interfaced with graphics are the Printronix MVP, the Envision 420, the Anadex 9620, the Okidata Microline 93, and the Data Products SPG-8010 and SPG-8050 printers. Like the plotter output, printer output can also be written to a disk file. When a printer is the output device, all of the data that represents a picture is accumulated in a buffer before the picture is actually printed. To accommodate the printer buffer, much more memory must be allocated for the picture file work area than when other output devices are used. In addition, since the actual printing does not begin until the picture is complete and the buffer is released, DisplayPicture, an operation that writes the entire picture, must be used. Other procedures that draw and display one vector or one text string, for example, instead of the whole picture will return an error code when the printer is the current output device.

SECTION 3
DEVICE-INDEPENDENT PROCEDURES

The device-independent procedures in the graphics library are used in application systems designed to run on any B 20 Graphics workstation. They permit output to hardcopy devices (such as plotters and dot matrix printers) as well as video display screens within B 21, B 22 and B 26 series.

The device-independent procedures are organized in this section by general function. The order in which procedures from the different groups are used in an application is very flexible and depends on the functional requirements of the application. The arrangement used in this guide follows a logical top-down graphics processing sequence. The 11 groups of procedures are shown in Table 3-1.

Table 3-1. Device-Independent Procedures by Function (Page 1 of 2)

Initialization	**Object**
ClearViewport	AddObject
InitGraphics	ClearLabels
SetLimits	ClearVectors
SetOutputDevice	CloseObject
SetOutputType	CloseTempObject
SetPlotterDevice	DisplayCurrentObject
SetPlotterMaterial	OpenTempObject
SetUserCoordinates	RemoveCurrentObject
	SetFirstObject
	SetNextObject
Picture	**Attribute**
AddPicture	SetColor
ClosePicture	SetCurrentPalette
DisplayPicture	SetDrawingMode
GetNumberOfObjects	SetLineType
OpenPicture	SetPictureColors
WritePicture	

Table 3-1. Device-Independent Procedures by Function (Page 2 of 2)

Drawing

Draw
DrawArc
DrawCircle
DrawLine
DrawRelative
FillRectangle
Move
MoveRelative

Text

SetCharacterSize
SetFont
SetLabelOrigin
WriteTextString

Font

GetFontName
GetFontNumber
GetNumberFonts
GetUserFontName
SetUserFont

Label

AddLabel
DeleteCurrentLabel
GetCurrentLabel
GetLabelData
ModifyLabel
SetFirstLabel
SetNextLabel

Transformation

GetTransformationData
SetScale
SetScaleRelative
SetTranslate
SetTranslateRelative

Viewing

GetWindowData
SetViewport
SetWindow

Cursor

GetCursorPosition
SetNDCCursorPosition
SetObjectCursorPos.
SetWorldCursorPosition
TurnCursorOff
TurnCursorOn

The following steps are presented as a guideline to illustrate typical use of the device-independent procedures. In this example, a picture is opened, an object is created, and the picture is saved in a picture file. The procedures, or examples of possible procedures, used to accomplish each step are included in parentheses.

1. Allocate memory for the picture file workarea (AllocMemorySL).

2. Initialize the graphics system (InitGraphics).

3. Open a new picture in write mode (OpenPicture).

4. Begin a new object and specify the range of user coordinates to be used for drawing the object (AddObject).

5. If the drawing is to be limited to a subset of the world coordinate system, specify the limits (SetLimits).

6. Use attribute, drawing, label, text, and font commands to create an object (for example, SetColor, DrawLine, Move, WriteTextString, AddLabel).

7. Close the object (CloseObject).

8. Save the picture (ClosePicture).

Transformation procedures can be used whenever an object is open. Viewing procedures can be used any time the picture is open.

This section contains a subsection for each group of device-independent procedures. The procedures within each group are ordered alphabetically. A brief description, the procedural interface, and the parameter definitions are included for each procedure. The conventions used for parameter names in graphics library procedures are defined in the B 20 Operating System Reference Manual. Besides the data type prefixes defined there, the graphics software introduces two new types: "r" and "w."

o r 4-byte short real number

o w word (16 bits)

In addition to describing the device-independent graphics library procedures, this section also includes a subsection on user-written procedures that can be called from the graphics software to provide extended capabilities for an application. A user-written procedure can be called, for example, from the graphics code that handles plotter output. An application can include procedures that are called by the graphics software to halt the plotter output while pens are changed or paper is loaded. User interaction through messages and replies on the video display unit can also be provided by user-written procedures.

INITIALIZATION PROCEDURES

The initialization procedures are used to set the values for different variables used by the graphics software.

There are eight initialization procedures:

- ClearViewport
- InitGraphics
- SetLimits
- SetOutputDevice
- SetOutputType
- SetPlotterDevice
- SetPlotterMaterial
- SetUserCoordinates

InitGraphics is always the first graphics function performed prior to using device-independent (high level) procedures. The other initialization procedures can be used to set their respective variables at any point.

ClearViewport

Description

ClearViewport clears the viewport. The video display screen is erased.

If a plotter or printer has been assigned as the output device, this procedure has no effect.

Procedural Interface

ClearViewport: ErcType

InitGraphics

Description

InitGraphics initializes the variables used by the graphics software. The display memory on the graphics control board is cleared, and the default line type and drawing mode values are set. The B 22 window defaults to the range (0,0) through (100,77.74), and the viewport is set to (0,0) through (1,.7744). On the B 21 series the range is (0,0) through (100,73.84), with the viewport set to (0,0) through (1.7384). The B 26 window defaults to the range (0,0) through (100,74), and the viewpoint is set to (0,0) through (1,.74).

InitGraphics must be the first graphics procedure called.

Procedural Interface

InitGraphics: ErcType

SetLimits

Description

SetLimits allows a portion of the world coordinate system to be defined as the area of interest. Used in conjunction with SetUserCoordinates, this procedure sets up a rectangular area, and SetUserCoordinates provides the range of user-defined coordinate values that are mapped to the rectangle.

This procedure could be used, for example, to define a box around a bar chart. If SetUserCoordinates is used, the user-defined coordinates supplied when the bar chart is drawn are mapped to the area defined by the box.

If SetLimits is not used, the default world coordinate area is the portion with the same aspect ratio as the video display screen; (0,0) to (100,77.74) for B 22, (0,0) to (100,73.84) for the B 21, and (0,0) to (100,74) for the B 26 series systems.

The order in which the SetLimits command is used is important. If called before an object is opened, it must be followed by an AddObject command or an OpenTempObject command. If used after an object is opened, it must be followed by a SetUserCoordinates command. If these conventions are not followed, the invocation of SetLimits will have no effect on subsequent drawing commands.

Procedural Interface

SetLimits (rXMin, rYMin, rXMax, rYMax): Erctype

where

rXMin	specifies the minimum X value in world coordinates.
rYMin	specifies the minimum Y value in world coordinates.
rXMax	specifies the maximum X value in world coordinates.
rYMax	specifies the maximum Y value in world coordinates.

SetOutputDevice

Description

SetOutput Device allows the output device to be reassigned. The default output device is the video display screen.

This procedure is used to assign a plotter or a dot matrix printer as the output device. Refer to the "User-Written Procedures" subsection below for information on application procedures that can be called by graphics library procedures to extend the capabilities for plotter output processing. Refer also to the descriptions of the following procedures: SetOutputType, SetPlotterDevice, and SetPlotterMaterial. These procedures should be called, if needed, before SetOutputDevice.

Procedural Interface

SetOutputDevice (iDevice): Erctype

where

iDevice	specifies the output device.
	0 = video display screen
	1 = plotter
	2 = dot matrix printer

SetOutputType

Description

SetOutputType specifies the code for the device that is to be used when the output is directed to a plotter or a printer. This procedure must be called before SetOutputDevice is called.

Procedural Interface

SetOutputType (iOutputType): ErcType

where

iOutputType	specifies the code for the output device.
	0 = HP7470A
	1 = HP7220C
	2 = Strobe 100
	3 = Printronix MVP
	4 = Anadex 9620
	5 = AP1351
	6 = B9253
	7 = Envision 420
	8 = not used
	9 = HP7475A
	10 = HP7220T
	11 = Okidata Microline 93
	12 = Data Products 8010

SetPlotterDevice

Description

SetPlotterDevice specifies either the name of the disk file where the output is to be written or, if the output is to be written directly to the output device, the configuration file for the device. If the output is not going to disk, the following type of configuration information is used:

```
           [COMM]A
HP7470A  - [COMM]B&[sys]<sys>PlotterConfig.sys
HP7220C

           [COMM]A
Strobe   - [COMM]B&[sys]<sys>StrobeConfig.sys

Anadex   - [LPT]
Printronix
```

In order to plot to a spooler or disk file, applications must declare an external byte variable called FSpool and assign it the value 255 (OFF hex) before calling SetPlotterDevice.

Procedural Interface

SetPlotterDevice (pbDevName, cbDevName): ErcType

where

pbDevName	
cbDevName	describe the disk file name or
configuration	file for the device.

SetPlotterMaterial

Description

SetPlotterMaterial specifies whether the output is to be plotted on paper or on a transparency. This procedure should be called before using SetOutputDevice. The SetOutputDevice initialization routine reduces the plotter speed when the output is going to be plotted on a transparency.

Procedural Interface

SetPlotterMaterial (iMaterial): ErcType

where

iMaterial	specifies whether the output is to be on
paper or	on a transparency.
	0 = paper
	1 = transparency

SetUserCoordinates

Description

SetUserCoordinates sets the user-defined coordinates used in the drawing procedures. The units supplied in this procedure are mapped to the world coordinate system. When user-defined coordinate positions are specified in subsequent procedures, the graphics software automatically translates the units to the world coordinate system.

SetLimits can be used in conjunction with this procedure to define a portion of the world coordinate system to which the user-defined coordinates are to be mapped.

Procedural Interface

SetUserCoordinates (rXMin, rYMin, rXMax, rYMax): Erctype

where

rXMin	specifies the minimum x value in user-defined coordinates to be mapped to the minimum X value in world coordinates.
rYMin	specifies the minimum Y value in user-defined coordinates to be mapped to the minimum Y value in world coordinates.
rXMax	specifies the maximum X value in user-defined coordinates to be mapped to the maximum X value in world coordinates.
rYMax	specifies the maximum Y value in user-defined coordinates to be mapped to the maximum Y value in world coordinates.

PICTURE PROCEDURES

The picture procedures are used to manage picture files and to manipulate pictures in display memory. When a new graphic representation is being created a picture is opened to save it. Likewise, when an object in an existing picture is to be modified, the first step is to open the picture. Once a picture is opened, the graphic representations within the picture can be be created, modified, and transformed. When the current picture has been fully processed, it is written to a picture file and closed. After the current picture is closed, another picture can be processed.

There are six picture procedures:

o AddPicture

o ClosePicture

o DisplayPicture

o GetNumberOfObjects

o OpenPicture

o WritePicture

OpenPicture must be performed before any of the other picture procedures can be used.

AddPicture

Description

AddPicture adds the specified picture file to the current picture. The added picture becomes part of the current picture. In B 21 and B 26 Color Graphics workstation applications, the fOverwritePalette parameter is used to specify which color palette should be used to draw the added object. TRUE = yes, overwrite with the palette from the added picture file. FALSE = no, use the palette that has already been set for the current picture.

Procedural Interface

```
AddPicture (pbPictureName, cbPictureName
   fOverwritePalette): ErcType
```

where

pbPictureName cbPictureName	describes the picture file to be merged into the current picture.

fOverwritePalette	specifies whether the palette from the added picture should overwrite the palette in the current picture.
	
	OFFh = TRUE, overwrite
	
	OOh = FALSE, do not overwrite

ClosePicture

Description

ClosePicture closes the current picture. If the parameter fSave is set to TRUE, the picture is written before it is closed. It is saved in the picture file previously specified in the OpenPicture command. The fSave parameter is set to FALSE when the picture has already been saved by a previous WritePicture command.

Procedural Interface

```
ClosePicture (fSave): ErcType
```

where

fSave	specifies whether the picture is to be written before it is closed.
	
	OFFh = TRUE, write.
	
	OOh = FALSE, do not write.

DisplayPicture

Description

DisplayPicture displays the current picture. It is used after OpenPicture to display a picture, and after a picture is modified, to redisplay it. The screen is not erased before the picture is displayed; the new information is merged or overlays parts of the existing picture. ClearViewport must be used before DisplayPicture if the screen is to be erased before displaying.

DisplayPicture calls the procedure, ReadInterruptKey to determine whether the output to the screen, plotter, or printer should be interrupted. The graphics library version of ReadInterruptKey returns a "0" status code which prompts DisplayPicture to continue writing the output without an interruption.

ReadInterruptKey can be replaced by a user-written procedure with the same name to halt the DisplayPicture process. Refer to the "User-Written Procedures" subsection for detailed information about the use of ReadInterruptKey.

SetPen is another procedure that is called by DisplayPicture and can be replaced by user-written code. When the output device is a plotter and DisplayPicture encounters a new pen number, SetPen is called. The purpose of SetPen is to enable the application to halt the plotter output and notify the user that the pen should be changed. Refer to the "User-Written Procedures" subsection for detailed information about the use of SetPen.

Procedural Interface

DisplayPicture (fInterruptOnKey): ErcType

where

fInterruptOnKey	indicates whether or not ReadInterruptKey is to be called. 0FFh = TRUE, ReadInterruptKey is called. 00h = FALSE, it is not.

GetNumberOfObjects

Description

GetNumberOfObjects returns the number of objects in the current picture. The number is placed in a 4-byte memory location.

Procedural Interface

GetNumberOfObjects (pNObjectsRet): ErcType

where

pNObjectsRet	points to the memory address where the number of objects in the picture is to be returned.

OpenPicture

Description

OpenPicture opens the specified picture. It is used to create new pictures and to modify existing ones.

One of three modes must be specified: read, write, or modify.

Read mode is used to view an existing picture. The size of the window or viewport can be changed, the objects within the picture can be transformed, but the objects cannot be modified.

Modify mode also requires an existing picture. This mode is used when new objects are to be added to the picture and when existing objects are to be modified.

Write mode is used to create a new picture. Objects can be created for the new picture or existing pictures can be added from picture files to create a complex picture. Write mode can also be used to open existing pictures from picture files. When write mode is used with an existing picture, the picture is deleted when the file is opened, and the new version replaces the old.

A segment of memory must be allocated before OpenPicture is executed. The memory area is used as a workarea and must be large enough to contain the whole picture. Refer to Appendix D for information about the memory requirements for pictures and objects.

OpenPicture must be called before SetOutputDevice when printing a picture and after SetOutputDevice when plotting a picture.

Procedural Interface

OpenPicture	(pbPictureName, cbPictureName, pbPassword, cbPassword, mode, pMemory, cParasMemory): Erctype

where

pbPictureName cbPictureName	describe a character string specifying the name of a picture file.
pbPassword cbPassword	describe the standard volume, directory, or file password that authorizes access to the picture file.
mode	is read (shared) or modify and write (exclusive). The mode is indicated by a 16-bit value representing the ASCII constants "mr" (mode read), "mm" (mode modify), or "mw" (mode write). In these ASCII constants, the first character (m) is the high-order byte and the second character (r, m, or w respectively) is the low-order byte. This is the reverse of the byte order for strings in B 20 programming languages.
pMemory cParasMemory	specifies the segment of memory to be used as a work area for the picture. The memory size is specified as the number of 16-byte paragraphs allocated.

WritePicture

Description

WritePicture writes the current picture to the picture file specified by pbPictureName, cbPictureName. If a picture file with this name already exists, it is overwritten by the current picture. When WritePicture is executed, the current picture is not closed; it remains the current picture, and processing can continue.

Procedural Interface

```
WritePicture (pbPictureName cbPictureName):
              ErcType
```

where

pbPictureName cbPictureName	specifies the picture file to which the current picture is to be written.

OBJECT PROCEDURES

The object procedures are used to add new objects to the current picture, and to modify existing objects. When there are multiple objects in a picture, only one can be processed at a time. There are also object procedures that are used to move through the objects in a picture to select the current object. When a current object is designated, subsequent commands operate on that object until another object is selected as the current object.

There are ten object procedures:

- AddObject
- ClearLabels
- ClearVectors
- CloseObject
- CloseTempObject
- DisplayCurrentObject
- OpenTempObject
- RemoveCurrentObject
- SetFirstObject
- SetNextObject

Before any object procedures can be used, a picture must be opened with OpenPicture, or the object must be declared as a temporary object by OpenTempObject.

AddObject

Description

AddObject is used to begin a new object that is to be part of the current picture. The object specified by pbObjectName, cbObjectName becomes the current object. All subsequent vector commands and labels are stored in this object's vector and label lists.

Minimum and maximum X and Y coordinates specify the range of user coordinates that the new object will use. User-defined coordinate values specified in a previous AddObject or SetUserCoordinate procedure will be overridden by this call.

A picture must be open in modify or write mode before AddObject can be used, and no other object may be open.

Procedural Interface

```
AddObject (pbObjectName, cbObjectName, rXMin,
           rYMin, rXMax, rYMax): ErcType
```

where

pbObjectName cbObjectName	specifies the name of theobject to be added. The maximum length for an object name is 12 characters.
rXMin	specifies the minimum X value of the object.
rYMin	specifies the minimum Y value of the object.
rXMax	specifies the maximum X value of the object.
rYMax	specifies the maximum Y value of the object.

ClearLabels

Description

ClearLabels clears the current object's label list. Since individual labels can be modified by ModifyLabel, this procedure is used only when all the labels are to be replaced.

A picture must be open in write or modify mode before ClearLabels can be used. An object must also have been designated as the current object.

Procedural Interface

```
ClearLabels: ErcType
```

ClearVectors

Description

ClearVectors clears the current object's vector list. Because individual vector commands cannot be modified, the whole list is cleared when an individual vector is to be recomputed.

A picture must be open in write or modify mode before ClearVectors can be used. An object must also have been designated as the current object.

Procedural Interface

```
ClearVectors: ErcType
```

CloseObject

Description

CloseObject closes the current object. An object must be closed before a new one can be selected as the current object.

Both a picture and an object must be open to use CloseObject. The object cannot be temporary.

Procedural Interface

```
CloseObject: ErcType
```

CloseTempObject

Description

CloseTempObject closes a temporary object.

An error condition occurs if there is not a temporary object open.

Procedural Interface

CloseTempObject: ErcType

DisplayCurrentObject

Description

DisplayCurrentObject displays the current object on the screen. The screen is not erased before the object is displayed. The current object is merged with the current contents of the screen.

A picture must be open before DisplayCurrentObject is used, and an object must have been designated as the current object.

Procedural Interface

DisplayCurrentObject: ErcType

OpenTempObject

Description

OpenTempObject opens a temporary object. When an object is temporary, subsequent commands are not saved in a picture file. If a picture has been opened, it must be closed before a temporary object can be opened. Likewise, if a current object has been designated, it must be closed before OpenTempObject can be used.

Minimum and maximum coordinates are specified to indicate the range of user coordinates that will be used for this object.User-defined coordinate values specified in a previous AddObject or SetUserCoordinates procedure will be overridden by this call.

Procedural Interface

```
OpenTempObject (rXMin, rYMin, rXMax, rYMax):
               ErcType
```

where

rXMin	specifies the minimum X value of the object.
rYMin	specifies the minimum Y value of the object.
rXMax	specifies the maximum X value of the object.
rYMax	specifies the maximum Y value of the object.

RemoveCurrentObject

Description

RemoveCurrentObject removes the current object from the current picture.

A picture must be open in write or modify mode, and an object must have been designated as the current object before RemoveCurrentObject can be used.

Procedural Interface

```
RemoveCurrentObject: ErcType
```

SetFirstObject

Description

SetFirstObject designates the first object in the picture as the current object. Objects are stored in the order they were created.

A picture must be open before SetFirstObject can be used.

Procedural Interface

SetFirstObject: ErcType

SetNextObject

Description

SetNextObject specifies a new current object. The object that follows the current object becomes the new current object. If a current object has not been designated when this procedure is called, then the first object in the picture becomes the current object. Objects are stored in the order they were created. If the current object is the last object in the picture when this procedure is called, the following error code is returned:

```
      ercEndOfObjects = 7628
```

The picture procedure, GetNumberOfObjects can be used in conjunction with SetNextObject to keep track of how many objects there are in the picture.

A picture must be open before SetNextObject can be used.

Procedural Interface

SetNextObject: ErcType

ATTRIBUTE PROCEDURES

Attribute procedures are used to set the values for attributes that are used in conjunction with drawing procedures. The attributes are line type, drawing mode, and color. Detailed information about these attributes is included in the "Drawing Attributes" subsection of Section 2, "Concepts."

Before an attribute procedure can be called, either a picture must be open in write or modify mode, and an object must be designated as the current object, or a temporary object must be open. The attribute procedures cannot be used if a printer has been assigned as the output device.

There are five attribute procedures:

- GetPictureColors
- SetColor
- SetCurrentPalette
- SetDrawing Mode
- SetLineType

GetPictureColors

Description

GetPictureColors returns the eight bytes that define the colors in the current palette. The color bytes are copied to the memory location specified by the parameter pRgbPaletteRet. Refer to Section 4, "Device-Dependent Procedures" for detailed information about color palettes.

Procedural Interface

GetPictureColors (pRgbPaletteRet): ErcType

where

pRgbPaletteRet	specifies the memory address ofthe buffer where the eight bytes used to define the current palette are to be returned.

SetColor

Description

SetColor specifies the color that is to be current. Subsequent drawing procedures will use the color designated by the parameter iColor. This parameter specifies the color in the current 8-byte color palette. The acceptable values are 1 through 8. Multicolor output to a video display unit is supported only on the B 21 and B 26 Color Graphics workstations.

When a B 22 video screen is the output device, the color attribute is ignored. When the output device is a plotter, the color attribute is interpreted as the pen number for the intended color. The user selects the colors and assigns a number for each pen.

Detailed information about the color attribute can be found in the "Drawing Attributes" subsection of Section 2, "Concepts" and in Section 4, "Device-Dependent Procedures."

Procedural Interface

SetColor (iColor): ErcType

where

iColor specifies the color to be used in subsequent commands, in the range 1 - 8. The default is 1.

SetCurrentPalette

Description

SetCurrentPalette is used in application systems designed for the B 21 and B 26 Color Graphics workstations. It specifies the eight bytes that define a new palette. The palette that is selected remains the current palette for subsequent procedures until SetCurrentPalette is called again. For detailed information about the use of color palettes, refer to the "Drawing Attributes" subsection of Section 2, "Concepts" and Section 4, "Device-Dependent Procedures."

If SetCurrentPalette is not called, the default color palette is used. The colors in the default palette are:

- red
- yellow
- green
- blue
- cyan
- magenta
- white
- black

Procedural Interface

SetCurrentPalette (pRgbPalette): ErcType

where

pRgbPalette	points to the memory address of the eight bytes that define the colors in the palette.

SetDrawingMode

Description

SetDrawingMode specifies the drawing mode that is to be current. The choices are set mode, clear mode, complement mode, and replace mode. Subsequent drawing procedures will use the drawing mode designated by the parameter iMode. Detailed information about the drawing modes, including an illustration, can be found in the "Drawing Attributes" subsection of Section 2, "Concepts."

The default is set mode.

Procedural Interface

SetDrawingMode (iMode): ErcType

where

iMode	specifies the drawing mode.

0 = Set Mode

1 = Clear Mode

2 = Complement Mode

3 = Replace Mode

SetLineType

Description

SetLineType specifies the line type that is to be current. Subsequent vector drawing procedures will use the line type designated by the parameter iLineType. A solid line is the default, and there are other patterns of dots and dashes. Detailed information about the line type attribute, including an illustration of the available line types can be found in the "Drawing Attributes" subsection of Section 2, "Concepts."

Procedural Interface

SetLineType (iLineType): ErcType

where

iLineType	specifies the line type to be used in subsequent drawing procedures. 0 - 7 = the standard line types.

DRAWING PROCEDURES

Drawing procedures are used to draw vectors, arcs, and circles, and to fill rectangles. When these drawing procedures are executed, the commands are saved in the vector list of the current object.

User-defined coordinate values are used in the X and Y parameters. The graphics software automatically translates the user-defined units to world coordinates. The limits of the coordinate system must be previously defined by SetUserCoordinates or AddObject before drawing procedures are used.

A picture must be open in write or modify mode (except when the object is temporary), and an object selected as the current object before a drawing procedure is used. An error message is returned if these procedures are used when the output device is a printer.

The Fill Rectangle procedure cannot be used with temporary objects.

There are eight drawing procedures:

- Draw
- DrawArc
- DrawCircle
- DrawLine
- DrawRelative
- FillRectangle
- Move
- MoveRelative

Draw

Description

Draw draws a vector from the current position to (rX,rY). The current color, line type and drawing mode are used. After the vector is drawn, the current position is set to (rX,rY). User-defined coordinate values are used in the parameters.

This command is saved in the vector list of the current object.

Procedural Interface

Draw (rX, rY): ErcType

where

rX specifies the X coordinate to which the line is to be drawn.

rY specifies the Y coordinate to which the line is to be drawn.

DrawArc

Description

DrawArc draws an arc by using the center position, radius, and angles provided in the parameters. User-defined coordinate values are used for these parameters. The angles are specified in radians from the center of the circle, and the arc is drawn in a counterclockwise direction. Figure 3-1 illustrates the drawing angles.

The current color, line type, and drawing mode are used. After the arc is drawn, the current position is set to the end of the arc.

This command is saved in the vector list for the current object.

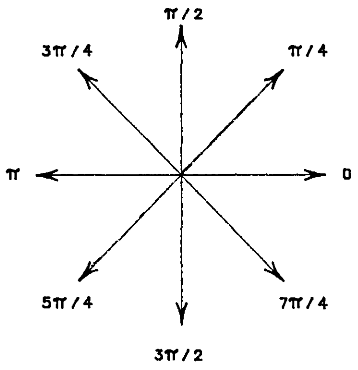


Figure 3-1. Angles in Radians

Procedural Interface

```
DrawArc (rXCenter, rYCenter, rsRadius, rAng1,
        rAng2): ErcType
```

where

rXCenter rYCenter	specify the position from which the angles are calculated to create the arc.
rsRadius	specifies the radius from the center point.
rAng1	specifies the angle used to set the beginning position of the arc.
rAng2	specifies the angle used to set the end position of the arc.

DrawCircle

Description

DrawCircle draws a circle with position rXCenter,rYCenter as the center and rSRadius as the radius. User-defined coordinate values are used for these parameters. After the circle is drawn, the zero-degree position on the circumference of the circle becomes the current position. The current line type, drawing mode, and color attributes are used with this procedure.

This command is saved in the vector list of the current object.

Procedural Interface

```
DrawCircle (rXCenter, rYCenter, rSRadius):
            ErcType
```

where

rXCenter rYCenter	specify the center of the circle.
rSRadius	specifies the radius of the circle.

DrawLine

Description

DrawLine draws a line by using the endpoints specified. The current color, line type, and drawing mode are used. After the line is drawn, the current position is set to (rX2,rY2). User-defined coordinate values are used in the parameters.

This command is saved in the vector list of the current object.

Procedural Interface

```
DrawLine (rX1, rY1, rX2, rY2): ErcType
```

where

rX1	specifies the X coordinate for the beginning of the line.
rY1	specifies the Y coordinate for the beginning of the line.
rX2	specifies the X coordinate for the end of the line.
rY2	specifies the Y coordinate for the end of the line.

DrawRelative

Description

DrawRelative draws a vector from the current position to the position offset by rDeltaX,rDeltaY.User-defined coordinate values are used in the parameters. The current color, line type and drawing mode are used. After the vector is drawn, the current position is set to (rX+rDeltaX,rY+rDeltaY).

This command is saved in the vector list of the current object.

Procedural Interface

DrawRelative (rDeltaX, rDeltaY): ErcType

where

rDeltaX	specifies the change in the X direction to reach the new position to which the line should be drawn.
rDeltaY	specifies the change in the Y direction to reach the new position to which the line should be drawn.

FillRectangle

Description

FillRectangle fills a rectangle with a pattern. There are six different patterns. Figure 3-2 illustrates the available patterns.

Procedural Interface

FillRectangle (rXMin, rYMin, rXMax, rYMax, bFillType): ErcType

where

rXMin rYMin rXMax rYMax	specify the lower left and upper right corners of the rectangle.
bFillType	specifies the fill pattern 0 - 5 = the fill patterns.

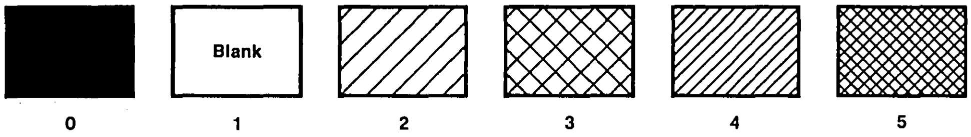


Figure 3-2. Fill Types

Move

Description

Move sets the current position. Subsequent drawing procedures will begin at this position. User-defined coordinate values are used in the parameters.

This command is saved in the vector list of the current object.

Procedural Interface

Move (rX, rY): ErcType

where

rX	specifies the X coordinate for the new current position.
rY	specifies the Y coordinate for the new current position.

MoveRelative

Description

MoveRelative sets the current position. The new current position is offset from the existing current position by rDeltaX and rDeltaY. Subsequent drawing procedures will begin at this position. User-defined coordinate values are used in the parameters.

This command is saved in the vector list of the current object.

Procedural Interface

MoveRelative (rDeltaX, rDeltaY): ErcType

where

rDeltaX	specifies the change in the X direction for the new current position.
rDeltaY	specifies the change in the Y direction for the new current position.

TEXT PROCEDURES

These procedures are used to create and modify text strings. Unlike labels, which are saved in the label list and can be modified, text strings are put in the vector list and cannot be modified. Text strings are used typically for text, such as units on axes and legends, that is not to be altered.

There are four text procedures:

- SetCharacterSize
- SetFont
- SetLabelOrigin
- WriteTextString

WriteTextString writes a text string in the current object, and the other procedures in this subsection set the attributes that are to be used when the text is drawn. A picture must be open in write mode, and a current object must have been designated before any of the text procedures can be used. The output device cannot be a printer. For detailed information about the attributes used with text, refer to the "Text Attributes" subsection of Section 2, "Concepts."

SetCharacterSize

Description

SetCharacterSize specifies the relative character size that will be used in subsequent WriteTextString procedures. Refer to the "Text Attributes" subsection in Section 2, "Concepts" for detailed information about the character size attribute.

Procedural Interface

SetCharacterSize (rSChars): ErcType

where

rSChars	specifies the character size used in subsequent calls to WriteTextString. 1 = standard (the default)

SetFont

Description

SetFont specifies the font that will be used in subsequent WriteTextString procedures. The graphics software contains four fonts: the default font, SimplexRoman, and three alternate fonts, ComplexRoman, DuplexRoman, and Gothic. Refer to the "Text Attributes" subsection in Section 2, "Concepts" for detailed information about the font attribute. Procedures for selecting fonts and establishing which fonts are available are found in the "Font Procedures" subsection of this section.

Procedural Interface

SetFont (pbFontName, cbFontName): ErcType

where

pbFontName cbFontName	specify the name of the font to be used in subsequent calls to WriteTextString.

SetLabelOrigin

Description

SetLabelOrigin specifies the current label origin to be used in subsequent WriteTextString procedures. The label origin is used to indicate how the text should be oriented in relation to the current position. Text can be placed left flush, right flush, or centered. These placements can be done at the top, middle, or bottom of the current position. Refer to the "Text Attributes" subsection of Section 2, "Concepts" for detailed information about the label origin attribute.

Procedural Interface

SetLabelOrigin (bLorg): ErcType

where

bLorg	specifies the label origin to be used in subsequent calls to WriteTextString.

0 = bottom left

1 = middle left

2 = top left

3 = bottom center

4 = middle center

5 = top center

6 = bottom right

7 = middle right

8 = top right

WriteTextString

Description

WriteTextString draws a text string in the current object. It uses the current text attributes and the current position. The current position can be set by Move or MoveRelative. The attributes that must be set before WriteTextString is executed are character size, label origin, and font. Refer to the "Text Attributes" subsection of section 2, "Concepts" for detailed information about text attributes. After the text string is written, the last position of the string becomes the current position.

This command and the text attributes are saved in the vector list of the current object.

Procedural Interface

```
WriteTextString (pbString, cbString): ErcType
```

where

pbString cbString	describe the string to be drawn at the current position.

FONT PROCEDURES

The graphics software contains four fonts. The name for each font that is used internally by the graphics software is called the internal name. Fonts can also have user-friendly names assigned. In addition, the file specification for all but SimplexRoman, which is the standard font, can be altered. The information required to use multiple fonts is kept in a file called Graphics.Fonts. This file allows applications to specify which fonts are used, where they are located, and what user-friendly names have been assigned to correspond to the internal names.

The syntax for entries in Graphics.Fonts is:

"user-friendly name":"internal name": file specification

where

"user-friendly name"	is the string that identifies the font in end-user transactions.
"internal name"	is the string that identifies the font internally in the graphics software. The internal names for the four fonts that are included with the graphics software are: SimplexRoman ComplexRoman DuplexRoman Gothic
file specification	specifies the location of the font.

Table 3-2 shows the Graphics.Font entries supplied with the graphics software.

Table 3-2. Graphics.Font Entries

Standard:	SimplexRoman:
Complex:	ComplexRoman: [SYS]<SYS>ComplexRoman.font
Bold:	DuplexRoman: [SYS]<SYS>DuplexRoman.font
Gothic:	Gothic: [SYS]<SYS>Gothic.font

There are five font procedures:

o GetFontName

o GetFontNumber

o GetNumberOfFonts

o GetUserFontName

o SetUserFont

The font procedures are used in conjunction with text procedures and label procedures to specify which font is to be used for alphanumeric strings.

GetFontName

Description

GetFontName returns the memory address and length of the internal name for a font. The requested font is specified by its index in the font description file, Graphics.Fonts.

GetNumberOfFonts can be called before GetFontName to determine the number of fonts in the file.

Procedural Interface

```
GetFontName (iFont, pPbFontName, pCbFontName):
            ErcType
```

where

iFont	specifies the number of the font in Graphics.Fonts. The acceptable values are 0 through (nFonts-1) where nFonts = the number of fonts in the file.
pPbFontName	points to the memory location where the address of the internal font name is to be returned.
pCbFontName	points to the memory location where the count of bytes in the internal font name is to be returned.

GetFontNumber

Description

GetFontNumber returns the index of the specified font in the font description file, Graphics.Fonts. The font is specified by its internal name.

Procedural Interface

```
GetFontNumber (pbFontName, cbFontName,
               piFontRet): ErcType
```

where

pbFontName cbFontName	specify an internal font name.
piFontRet	points to the memory location where the font number is to be returned.

GetNumberOfFonts

Description

GetNumberOfFonts returns the number of font entries in Graphics.Fonts to indicate how many fonts are available for the application.

Procedural Interface

```
GetNumberOfFonts (pnFontsRet):  ErcType
```

where

pnFontsRet	points to the memory location of the word where the number of fonts is returned.

GetUserFontName

Description

GetUserFontName returns the memory address and length of the user-friendly name for a font that is specified by its index in the font description file, Graphics.Fonts.

GetNumberOfFonts can be called before GetUserFontName to determine the number of fonts in the file.

Procedural Interface

```
GetUserFontName (iFont, pPbFontName,
                 pCbFontName): ErcType
```

where

iFont	specifies the number of the font in Graphics.Fonts. The acceptable values are 0 through (nFonts-1) where nFonts = the number of fonts in the file.
pPbFontName	points to the memory location where the address of the user-friendly font name is to be returned.
pCbFontName	points to the memory location where the count of bytes in the user-friendly name is to be returned.

SetUserFont

Description

SetUserFont sets the current font by specifying the user-friendly name for the font. The user-friendly name is translated to the corresponding internal name. The internal name is then stored as the font attribute in the vector list during subsequent WriteTextString procedures.

Procedural Interface

```
SetUserFont (pbFontName, cbFontName,): ErcType
```

where

pbFontName cbFontName	specify the user-friendly name of the font.

LABEL PROCEDURES

The label procedures are used to create labels to accompany the vector portion of an object. Label procedures are also used to modify existing labels in the current object's label list. Before a label procedure can be used, a picture must be open in write or modify mode and an object designated as the current object.

Coordinate parameters in the label procedures must have world coordinate values.

When an existing label is selected to be modified, it is copied into a workarea structure. After the modifications are made in the workarea, the new label is written back into the object to replace the current label. The coordinate positions, text, and attributes of the label are saved in the label list.

Table 3-3 shows the format of the label structure.

There are seven label procedures:

- AddLabel
- DeleteCurrentLabel
- GetCurrentLabel
- GetLabelData
- ModifyLabel
- SetFirstLabel
- SetNextLabel

Table 3-3. Label Structure

ITEM	SIZE	CONTENTS
rXStart	real	the starting X position for the label
rYStart	real	the starting Y position for the label
rXLowRet	real	the leftmost X position of the label, computed by the label procedures
rYLowRet	real	the lowest Y position of the label, computed by the label procedures
rXHighRet	real	the rightmost X position of the label, computed by the label procedures
rYHighRet	real	the highest Y position of the label, computed by the label procedures
rsChars	real	the size of the label's characters
bLorg	byte	the label origin of the label

Table 3-3. Label Structure (Cont.)

ITEM	SIZE	CONTENTS
bPen	byte	the pen number for the label for plotter output, or in B 21 series Color Graphics workstation applications, the color number (1-8)
bUserID	byte	the unique identifier of the label
fPositionSet	byte	a value that can be set by the application to indicate the label position has been set
cbFontName	byte	the number of bytes in the internal name of the font for this label
rgbFontName(12)	byte	the string that describes the font for this label
rgbReserved(20)	byte	reserved
cbLabel	byte	the number of bytes in the label text
rgbLabel(cbLabel)	byte	the actual text of the label, which is cbLabel bytes long

AddLabel

Description

AddLabel adds a new label to the current object at the position specified. The parameter values for the coordinates must be world coordinate system values. The label is added to the label list of the current object.

There are three label attributes that are set by parameter entries in this procedure: character size, label origin, and font name. Character size specifies the scale of the characters in the label. The label origin indicates how the label is to be oriented in relation to the current position in the world coordinate system. The label can be positioned horizontally to the right, left, or center, and vertically to the top, middle, or bottom. The font attribute indicates the font that is to be used for the label text. Presently, there are four fonts provided by the graphics software. A unique identification is also provided for the label. In future modification processes, this identification can be used to select the proper label from the label list.

Detailed information about all three text attributes can be found in the "Text Attributes" subsection of Section 2, "Concepts." Additional information about the fonts that are available and how they are used can be found in the "Fonts Procedures" subsection above.

If the label has a zero length, an error condition occurs. Also, an error condition occurs if a printer has been assigned as the output device when this command is executed.

Procedural Interface

```
AddLabel (rX, rY, pbString, cbString, rSChars,
          bLorg, bPen, bUserID, pbFontName,
          cbFontName): ErcType
```

where

rX rY	specify the position of the label.
pbString cbString	specify the text of the label.
rSChars	specifies the character size of the label.
bLorg	specifies the label origin of the label.
bPen	specifies the pen number for plotter output, or in B 21 series Color Graphics workstation applications, the color number.
bUserID	specifies the label identifier.
pbFontName cbFontName	specify the internal name of the font to be used for the label.

DeleteCurrentLabel

Description

DeleteCurrentLabel erases the current label from the display screen and removes the label from the current object's label list. A label must be designated as the current label before DeleteCurrentObject is used. The output cannot be a printer when this command is executed.

Procedural Interface

```
DeleteCurrentLabel: ErcType
```

GetCurrentLabel

Description

GetCurrentLabel copies the current label from the current object's label list into a workarea structure where it can then be modified. A segment of memory must be allocated for the workarea. Refer to the introduction of this subsection, "Label Procedures", for information about the label structure. A label must be designated as the current label before GetCurrentLabel is used.

Procedural Interface

GetCurrentLabel (pLabelRet, sLabelRet): ErcType

where

pLabelRet	points to a structure where the label is to be copied.
sLabelRet	specifies the maximum size of the structure.

GetLabelData

Description

GetLabelData computes and fills in the boundaries of a label located in a workarea structure. Refer to the introduction to this subsection, "Label Procedures" for information about the label structure, including which parameters have values returned by GetLabelData.

This procedure is used to determine if a new label will fit, as is, in the world coordinate system. If it does not fit, the following error message is returned:

ercCharOutOfBounds: 7644

Procedural Interface

GetLabelData (pLabelRet): ErcType

where

pLabelRet	points to the structure containing the label.

ModifyLabel

Description

ModifyLabel replaces the current label with the label specified by the parameter pModifiedLabel. The current label in the object's label list is replaced by a new label that is located in a workarea structure. The structure contains the label text and attributes. Refer to the introduction of this subsection, "Label Procedures" for information about the label structure.

This procedure is used in conjunction with GetCurrentLabel, SetFirstLabel, or SetNextLabel. A label must be designated as the current label by one of these procedures before ModifyLabel can be used. These procedures copy an existing label into a workarea. After the label is modified in the workarea, it is written back into the current object's label list by ModifyLabel. The modified label is also displayed in the picture. The output device cannot be a printer when ModifyLabel is executed.

If the bUserID parameter in AddLabel was used to assign a unique identification when the label was created, this identification can be used in the modification process to quickly locate the label that is to be modified.

Procedural Interface

ModifyLabel (pModifiedLabel): ErcType

where

pModifiedLabel	points to the structure that contains the modified label.

SetFirstLabel

Description

SetFirstLabel selects the first label from the current object's label list and makes it the current label. The label is moved to a workarea where it can then be modified.

A segment of memory must be allocated as a workarea for the label structure. Refer to the introduction to this subsection, "Label Procedures", for information about the label structure.

Procedural Interface

SetFirstLabel (pLabelRet, sLabelRet): ErcType

where

pLabelRet	points to a structure where the label is to be copied.
sLabelRet	specifies the maximum size of the structure.

SetNextLabel

Description

SetNextLabel selects the next label from the current object's label list and makes it the current label. If a label has not been previously selected as the current label when this procedure is used, then the first label becomes the current label. If the current label is the last label in the label list, the following error message is returned, because there is no "next" label:

    ercEndOfLabelList = 7641

The selected label is copied into a workarea structure where it can be modified. A segment of memory must be allocated for the label structure. Refer to the introduction to this subsection, "Label Procedures", for information about the label structure.

Procedural Interface

SetNextLabel (pLabelRet, sLabelRet): ErcType

where

pLabelRet	points to a structure where the label is to be copied.
sLabelRet	specifies the maximum size of the structure.

TRANSFORMATION PROCEDURES

The transformation procedures are used to translate and scale the current object so that its size, shape, or position is altered on the display screen. The translation factors and the scalar units are stored in the transformation list of the current object. When the transformation procedures are performed, the original commands and labels remain, unmodified, in the vector and label lists for the object. Thus, when a translated object is redrawn or written to the display area from a picture file, the object is drawn by first executing the vector list commands for the full-size object. Then, the object is transformed by using the translation and scalar units in the transformation list. If the object is translated or scaled again, the new units replace the existing ones in the transformation list.

An object must be designated as the current object before the transformation procedures can be used.

There are five transformation procedures:

o GetTransformationData

o SetScale

o SetScaleRelative

o SetTranslate

o SetTranslateRelative

GetTransformationData

Description

GetTransformationData returns the transformation values for the current object. The values are returned in the following format:

```
     rXScale      (4 bytes)
     rXTranslate  (4 bytes)
     rYScale      (4 bytes)
     rYTranslate  (4 bytes)
```

Procedural Interface

GetTransformationData (pTransformRet): ErcType

where

pTransformRet	points to the memory address where the current object's transformation values are to be returned.

SetScale

Description

SetScale is used to change the size or shape of the current object. It scales the current object from its full size by the factors supplied in the parameters. If the scaling causes any part of the object to be outside the world coordinate area, the following error message is returned:

    ercBadTransformationParameter = 7647

The parameters, rXScale and rYScale, are real numbers between 0 and 1. These units are saved in the transformation list of the current object.

Procedural Interface

SetScale (rXScale, rYScale): ErcType

where

rXScale	specifies the scaling of the object in the X direction.
rYScale	specifies the scaling of the object in the Y direction.

SetScaleRelative

Description

SetScaleRelative is used after SetScale to scale the object farther from its original size or shape. This procedure scales the current object in the X direction by the current X scale factor plus the relative X scale factor supplied in the parameter. Likewise, the Y direction is scaled by the current Y scale factor plus the relative Y scale factor supplied in the parameter. If the scaling causes any part of the object to be outside the world coordinate area, the following error message is returned:

```
     ercBadTransformationParameter = 7647
```

The scaling units are saved in the transformation list of the current object.

Procedural Interface

```
SetScaleRelative (rXScaleRelative,
                  rYScaleRelative): ErcType
```

where

rXScaleRelative	specifies the scaling of the object relative to the current scale in the X direction.
rYScaleRelative	specifies the scaling of the object relative to the current scale in the Y direction.

SetTranslate

Description

SetTranslate is used to move the current object to another position in the current picture. This procedure translates the current object from (0,0), the lower left corner of the world coordinate area. The X and Y factors supplied in the parameters are used to determine the new position. Objects should be reduced by SetScale before they are translated. If either of the translation factors causes part of the object to be outside the world coordinate area, the following error message is returned.

ercBadTransformationParameter = 7647

The X and Y unit parameters must be specified in world coordinate system values. These translation units are saved in the transformation list of the current object.

Procedural Interface

SetTranslate (rXTranslate, rYTranslate): ErcType

where

rXTranslate	specifies the translation in the X direction.
rYTranslate	specifies the translation in the Y direction.

SetTranslateRelative

Description

SetTranslateRelative translates the current object from its current position by the X and Y factors supplied in the parameters. If either of the translation factors cause part of the object to be outside the world coordinate area, the following error message is returned:

```
ercBadTransformationParameter = 7647
```

The X and Y unit parameters must be specified in world coordinate system values.

Procedural Interface

```
SetTranslateRelative (rXTranslateRelative, rYTranslateRelative):
                     ErcType
```

where

rXTranslateRelative
: specifies the translation relative to the current translation in the X direction.

rYTranslateRelative
: specifies the translation relative to the current translation in the Y direction.

VIEWING PROCEDURES

The viewing procedures alter the perspective from which the current picture is viewed. By reducing the size of the window and focusing on just a small part of the picture, for example, the selected portion will be expanded to fit the whole viewport. A large picture can be panned by moving a small window from one position to another. The shape of the window can also be changed to alter the aspect ratio.

The viewport can be modified to define a smaller portion of the display area. The position or shape of the output display can also be altered.

The viewing procedures operate on pictures, not objects, and therefore, have no effect on the structure of the objects within the picture. The vector list, label list and transformation values for each object within the current picture are unchanged.

There are three viewing procedures:

o GetWindowData

o SetViewport

o SetWindow

GetWindowData

Description

GetWindowData returns the lower left and upper right corners of current window. The values returned are world coordinate units.

Procedural Interface

GetWindowData (pWindowData): ErcType

where

pWindowData	specifies the memory address where the coordinate values for the current window are returned. The format for the returned values is:

rXMin rYMin	the coordinates of the lower left corner of the window.
rXMax rYMax	the coordinates of the upper right corner of the window.

SetViewport

Description

SetViewport defines the portion of the video display screen that is to be used for the viewport. The default size is the entire screen. Frequently some portion of the display screen is needed for messages or forms. This is one instance when the size of the viewport should be reduced to a smaller portion of the screen.

Because the viewport defines the output device display area, the coordinates supplied in the parameters are normalized device coordinate values.

Procedural Interface

```
SetViewport (rXMin, rYMin, rXMax, rYMax):
             ErcType
```

where

rXMin	specifies the left edge of the viewport.
rYmin	specifies the bottom edge of the viewport.
rXMax	specifies the right edge of the viewport.
rYMax	specifies the top edge of the viewport.

SetWindow

Description

SetWindow defines the portion of the picture that is to be projected onto the viewport. The parameters define the dimensions of the window, and they must be entered as world coordinate system values. By changing the window, it is possible to zoom in and out on a portion of the picture, pan across the picture, and change the aspect ratio of the picture in relation to the screen.

Procedural Interface

```
SetWindow (rXMin, rYMin, rXMax, rYMax): ErcType
```

where

rXMin	specifies the left edge of the window.
rYMin	specifies the bottom edge of the window.
rXMax	specifies the right edge of the window.
rYMax	specifies the top edge of the window.

CURSOR PROCEDURES

The cursor procedures are used to position the cursor on the display screen. The cursor can be set to refer to an object, a picture, or the whole viewport.

There are six cursor procedures:

- GetCursorPosition
- SetNDCCursorPosition
- SetObjectCursorPosition
- SetWorldCursorPosition
- TurnOffCursor
- TurnOnCursor

GetCursorPosition

Description

GetCursorPosition returns the position of the cursor. The cursor's position is described in all the cursor positioning units, that is, by normalized device units for the screen position and by world coordinate units for the picture and object positions.

Procedural Interface

GetCursorPosition (pCursorStatusRet): ErcType

where

pCursorStatusRet	points to the memory address where the current cursor position is returned in the following format:

rXNDC	(4 bytes)
rYNDC	(4 bytes)
rXWorld	(4 bytes)
rYWorld	(4 bytes)
rXObject	(4 bytes)
rYObject	(4 bytes)

SetNDCCursorPosition

Description

SetNDCCursorPosition is used to position the cursor anywhere within the viewport. It moves the cursor to a position defined by normalized device coordinates. The cursor direction is also set. If the cursor is already visible on the screen when this procedure is used, then the old cursor is erased when the new one is drawn.

Procedural Interface

SetNDCCursorPosition (rX, rY, bDir): ErcType

where

rX	specifies the X coordinate for the cursor position.
rY	specifies the Y coordinte for the cursor position.
bDir	specifies the direction of the cursor arrow. 0 = up 1 = down 2 = right 3 = left

SetObjectCursorPosition

Description

SetObjectCursorPosition moves the cursor to a position within the current object. The position is specified with world coordinate system values and describes where the cursor is to be in the full-size object. If the object is transformed, the cursor position is adjusted to remain in the same relative position specified for the full-size object. The direction of the cursor is also included.

Procedural Interface

SetObjectCursorPosition (rX, rY, bDir): ErcType

where

rX	specifies the X coordinate for the cursor in the current object.
rY	specifies the Y coordinate for the cursor in the current object.
bDir	specifies the direction of the cursor arrow. 0 = up 1 = down 2 = right 3 = left

SetWorldCursorPosition

Description

SetWorldCursorPosition moves the cursor to a position within the current picture. The cursor is independent of any objects on the screen and is, therefore, not moved when an object is transformed. The position is specified with world coordinate values, and the direction of the cursor is also included.

Procedural Interface

SetWorldCursorPosition (rX, rY, bDir): ErcType

where

rX	specifies the X coordinate for the cursor in the picture.
rY	specifies the Y coordinate for the cursor in the picture.
bDir	specifies the direction of the cursor arrow.

0 = up

1 = down

2 = right

3 = left

TurnOffCursor

Description

TurnOffCursor turns off the cursor.

Procedural Interface

TurnOffCursor: ErcType

TurnOnCursor

Description

TurnOnCursor displays the cursor at its current location.

Procedural Interface

TurnOnCursor: ErcType

USER-WRITTEN PROCEDURES

In addition to the device-independent procedures discussed above, the graphics library also contains procedures that are called from within the graphics software. The graphics library versions of these procedures are defaults and can be replaced by user-written code. These called procedures are provided to allow application programs to include special processing capabilites within the device-independent procedures.

Calls to user-written procedures are made from within DisplayPicture, for example, to allow applications to interrupt the plotter output and change pens or load paper. There are three graphics library procedures that can be replaced with user-written versions:

- LoadPaper
- ReadInterruptKey
- SetPen

LoadPaper

Description

LoadPaper enables an application program to display a message on the screen to tell the user to load a piece of paper in the plotter. The default LoadPaper procedure returns without displaying a message.

This procedure is called after the plotter has been initialized by SetOutputDevice.

Procedural Interface

```
LoadPaper (fPaper):  ErcType
```

where

fPaper	indicates whether the output is to be paper or a transparency. 0FFh = TRUE, the output is paper. 00h = FALSE, the output is transparency.

ReadInterruptKey

Description

ReadInterruptKey enables an application to interrupt the process of displaying a picture. This procedure is called from within DisplayPicture if the parameter fInterruptOnKey supplied to DisplayPicture is set to TRUE. The default version of ReadInterruptKey returns a status code of zero. DisplayPicture then continues to display the output. To cause an interrupt, any nonzero code can be returned.

Procedural Interface

```
ReadInterruptKey: ErcType
```

SetPen

Description

SetPen enables an application to halt the plotter output while the user changes one of the pens. Messages on the video display to instruct the end user can be included. This procedure is called from within the DisplayPicture procedure when the plotter is the output device and a color attribute is encountered. SetPen is called to compare the color with the numbers of the pens that are loaded in the plotter. If the correct pen is already loaded, SetPen sets a "change pen" flag to FALSE. Processing continues without notifying the user to change the pen. The DisplayPicture procedure knows from the flag that the pen does not have to be changed.

If the specified pen is not loaded, the change pen flag is set to TRUE, the plotter output halted, and the user notified to change the pen. The pen number should be set to 1 if the new pen is in the left pen holder and 2, if the new pen goes on the right. Another internal procedure, SelectPen, is called for this purpose. SelectPen returns the pen that is to be changed to its holder. SelectPen is called with one parameter set to zero before the plotter output is interrupted.

SetPen, as described here, is the default version. Applications designers can modify or replace the default SetPen process with another version.

Procedural Interface

SetPen (piColor, pfChangePen): ErcType

where

piColor	points to the memory address of the pen number.
pfChangePen	points to the memory address of the change pen flag.

SECTION 4
DEVICE-DEPENDENT PROCEDURES

For applications where transportability is not an issue, the device-dependent procedures can be used for high-speed graphics. The device-dependent procedures in the graphics library can be used only on graphics workstations containing the appropriate graphics control board. The graphics control board display memory is mapped using the coordinate positions of display screens within the following workstations:

* B 21 (color)
* B 22 (monochrome)
* B 26 (color and monochrome)

Most of the device-dependent procedures can be used in B 21, B 22, and B 26 applications; however, running an application system that uses these procedures limits you to the type of workstation for which it was designed. The procedures supported on these Graphics workstations are discussed first. Then, the procedures which are dependent upon the B 21 and B 26 color workstations are discussed.

The device-dependent procedures are grouped by function into four categories. Table 4-1 shows the four groups, with the procedures listed alphabetically.

Table 4-1. Device-Dependent Procedures by Function

Control	Vector and Arc Manipulation
ClearScreen	ClearScreenRectangle
InitScreenGraphics	DrawScreenArc
SetCommandScreen	DrawScreenLine
SetVisibleScreen	FillScreenRectangle
TurnOffGraphics	LoadSoftPattern
TurnOffGraphicsColor	SetScreenDrawingMode
TurnOnGraphics	SetScreenLineType
TurnOnGraphicsColor	
Color	**Alphanumeric Attribute**
LoadColor	LoadColorStyleRam
LoadColorMapper	SetStyleRamEntry
SetColorMapper	SetStyleRam

CONTROL PROCEDURES

These procedures handle the screen display control functions. Two such functions are setting the visible display memory plane and the current plane. Control procedures are also used to display the visible screen and to clear the video display screen.

There are eight control procedures:

- o ClearScreen
- o InitScreenGraphics
- o SetCommandScreen
- o SetVisibleScreen
- o TurnOffGraphics
- o TurnOffGraphicsColor
- o TurnOnGraphics
- o TurnOnGraphicsColor

ClearScreen

Description

ClearScreen clears the current screen and the display plane.

Procedural Interface

ClearScreen: ErcType

InitScreenGraphics

Description

InitScreenGraphics clears the display memory and resets the default values for the line type, drawing mode, visible screen and command screen. This procedure also sets a flag that indicates to the B 20 Executive, when it is reloaded, that the initial command screen must be redisplayed.

Procedural Interface

InitScreenGraphics: ErcType

SetCommandScreen

Description

SetCommandScreen is used only in B 22 Graphics workstation applications. It specifies which screen, or display memory plane, is to be current. Subsequent operations, such as drawing commands, that affect the display memory will use the screen specified.

Procedural Interface

SetCommandScreen (iScreen): ErcType

where

iScreen	specifies which of the two display memory planes is to be used for the current screen.
	0 = the first plane
	1 = the second plane

SetVisibleScreen

Description

SetVisible Screen is used only in B 22 Graphics workstation applications. It specifies which screen, or display memory plane, is to be the visible screen. The visible screen is the one that can be displayed by TurnOnGraphics. The visible screen and the current screen are independent of each other. One display memory plane may be both visible and current simultaneously, or not, depending on the subsequent functions to be performed.

Procedural Interface

SetVisibleScreen (iScreen): ErcType

where

iScreen	specifies which of the two display memory planes is to be used for the visible screen. 0 = the first plane 1 = the second plane

TurnOffGraphics

Description

TurnOffGraphics turns off the video display screen. Unlike ClearScreen, it does not erase the visible screen in display memory.

Procedural Interface

TurnOffGraphics: ErcType

TurnOffGraphicsColor

Description

TurnOffGraphicsColor sets a color display to monochrome, regardless of any selected pens or color mapper settings.

Procedural Interface

TurnOffGraphicsColor: ErcType

TurnOnGraphics

Description

TurnOnGraphics displays the screen that has been set as the visible screen.

Procedural Interface

```
TurnOnGraphics:  ErcType
```

TurnOnGraphicsColor

Description

TurnOnGraphicsColor restores colors on the display and returns a color workstation to color graphics.

Procedural Interface

```
TurnOffGraphicsColor:  ErcType
```

VECTOR AND ARC MANIPULATION PROCEDURES

These procedures are used to draw vectors and arcs on the current screen. Rectangular areas can also be filled and cleared. The line types and drawing modes used in the high-level procedures can also be specified in the device-dependent procedures. In addition, there is a procedure to load a user-defined halftone pattern as an alternative line type.

For detailed information about line type and drawing mode options, refer to the "Drawing Attributes" subsection of Section 2, "Concepts".

Vectors and arcs are plotted by calculating a line between the specified endpoints. In B 22 Graphics workstation applications, the display memory is mapped using coordinate positions for a screen resolution of 656-by-510 pixels. Coordinate position (0,0) is the lower left corner of the screen, and coordinate position (655,509) is the upper right corner. In B 21 Color Graphics workstation applications, the coordinate positions range from (0,0) in the lower left corner to (431,318) at the upper right.

The B 26 display memory has a screen resolution of 718-by-348 pixels. However, coordinate position (4,0) is the lower left corner (1435,1043) and is the top right. There are two coordinate positions per pixel horizontally and three coordinate positions per pixel vertically on a B 26.

There are seven vector and arc manipulation procedures:

- o ClearScreenRectangle
- o DrawScreenArc
- o DrawScreenLine
- o FillScreenRectangle
- o LoadSoftPattern
- o SetScreenDrawingMode
- o SetScreenLineType

ClearScreenRectangle

Description

ClearScreenRectangle clears a rectangular area of display memory. The coordinates for the lower left corner of the rectangle are entered as 16-bit words. The height and width of the rectangle are also entered as words.

Procedural Interface

```
ClearScreenRectangle (wXStart, wYStart, wWidth, wHeight):
                      ErcType
```

where

wXStart wYStart	specify the lower left corner of the rectangular area.
wWidth	specifies the width of the rectangle.
wHeight	specifies the height of the rectangle.

DrawScreenArc

Description

DrawScreenArc draws an arc on the current screen. There are parameters to specify the start of curvature, the drawing direction, the radius, and the sines for the endpoints.

Figure 4-1 illustrates the values that are used for the parameters in DrawScreenArc.

The coordinate position for the start is the actual screen position. The X and Y values are specified as 16-bit words.

The maximum size arc that can be drawn by this operation is an octant of a circle. To draw a larger arc, DrawScreenArc must be called again until the intended size is reached. Figure 4-2 illustrates the drawing directions.

On a B 26, DrawScreenArc draws arcs using actual pixel coordinates rather than the adjusted screen coordinates. For this reason, arcs are stretched vertically, since there are more pixels per inch horizontally than vertically.

Procedural Interface

DrawScreenArc (wX, wY, wDir, wD, wD2, wDC, wDM): ErcType

where

wX wY	specify the coordinate position for the start of the arc.
wDir	specifies the drawing direction.
wD	specifies the radius - 1, in pixels.
wD2	specifies 2 * (radius - 1), in pixels.
wDC	specifies radius * sine(phi), rounded up, where the phi = the angle between the axis and the far end of the arc: $wDC < \frac{radius}{\sqrt{2}} \quad 0 < phi <= \pi/4$
wDM	specifies radius * sine(theta), rounded down, where theta = the angle between the axis and the start of the arc: 0<wDM<wDC $0 <= theta < \pi/4$

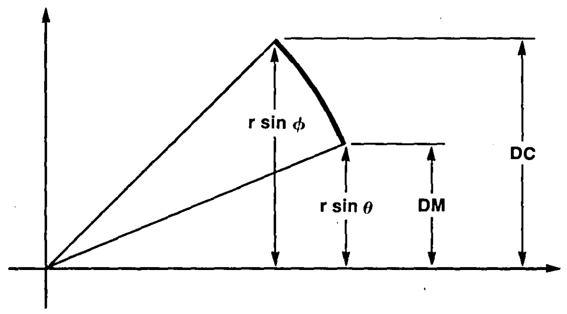


Figure 4-1. Determining Arc Length

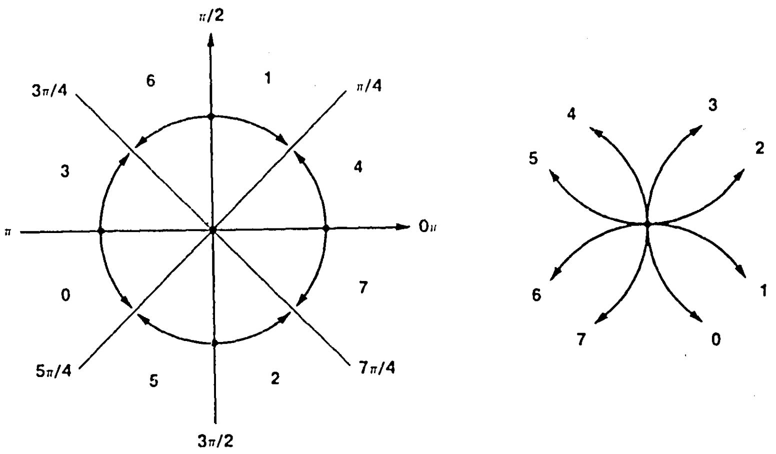


Figure 4-2. Drawing Directions (Angles in Radians)

DrawScreenLine

Description

DrawScreenLine draws a vector on the current screen. The coordinates entered as parameters are used for the endpoints. The vector bits are calculated according to the current line type. The bits in the line are then compared to the existing display memory bits and written according to the drawing mode.

The coordinates specify the actual screen positions for the endpoints, and they are entered as 16-bit words.

Procedural Interface

DrawScreenLine (wX1, wY1, wX2, wY2): ErcType

where

wX1 wY1	specify the coordinate position of the beginning of the vector.
wX2 wY2	specify the coordinate position of the end of the vector.

FillScreenRectangle

Description

FillScreenRectangle fills a rectangle in display memory with a pattern. The values for the coordinates that define the rectangle are entered as 16-bit words. The pattern is also specified as a parameter.

Procedural Interface

```
FillScreenRectangle (wX1, wY1, wX2, wY2, bFillType):  ErcType
```

where

wX1 wY1	specify the lower left corner of the rectangle.
wX2 wY2	specify the upper right corner of the rectangle.
bFillType	specifies the pattern that is to be used to fill the rectangle. Figure 4-3 illustrates the patterns.

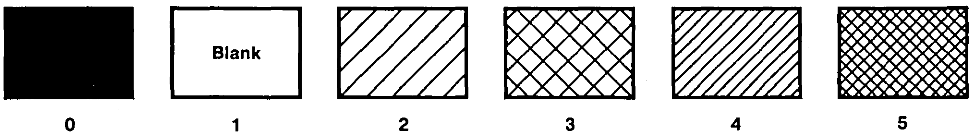


Figure 4-3. Fill Types

LoadSoftPattern

Description

LoadSoftPattern provides for the use of user-defined line type. This procedure defines one single line type. Besides a parameter to identify the line type, there is a word to hold the 16-bit pattern itself. To use a line type that has been defined with this procedure, the value for the line type parameter in SetScreenType must be within the range of 8 to 15. The parameter value minus the high-order 8 bit is the index to the soft pattern.

Procedural Interface

LoadSoftPattern (iPattern, wPattern): ErcType

where

iPattern	specifies the line type. The acceptable values are 0-7.
wPattern	specifies the 16-bit pattern.

SetScreenDrawing Mode

Description

SetScreenDrawingMode specifies the drawing mode that will be used in subsequent vector and arc drawing operations. There are four modes: set mode, clear mode, complement mode, and replace mode. Refer to the "Drawing Attributes" subsection in Section 2, "Concepts" for detailed information about drawing modes.

Procedural Interface

SetScreenDrawingMode (iMode): ErcType

where

iMode specifies the drawing mode to be set.

0 = set

1 = clear

2 = complement

3 = replace

SetScreenLineType

Description

SetScreenType specifies the dot pattern to be used for the line when vectors are drawn. There are eight line types. A solid line is the default, and there are other combinations of dots and dashes. This procedure may also be used in conjunction with LoadSoftPattern to specify a user-defined line type. Refer to the "Drawing Attributes" subsection in Section 2, "Concepts" for detailed information about line types.

Procedural Interface

SetScreenLineType (iLineType): ErcType

where

iLineType	specifies the line type.
	0 - 7 = the standard line types (see figure 2-1)
	8 - 15 = reserved for user-defined line types (see LoadSoftPattern)

COLOR PROCEDURES

The color procedures are available only on the B 21 and B 26 Color Graphics workstations. The colors that appear on the video displays for these workstations are defined by how much red, green, and blue they contain. There are 64 different combinations of these three primary colors, and any eight of these 64 possibilities can be displayed on the screen at one time.

An 8-byte memory work area is used to specify the eight colors and load the color mapper (on the B 21 graphics control board or the B 26 graphics controller module). The set of eight colors used by the graphics software is called the color palette.

Each byte in the color palette specifies one color. The low-order six bits of each byte are used as 2-bit color settings for red, green, and blue, respectively. Each 2-bit entry defines the intensity of the primary color it represents. The composition of the color is derived from the three 2-bit settings combined together. The bit settings correspond to color intensity as follows:

00	= none of this color is present
01	= a low intensity of this color is present
10	= a medium intensity of this color is present
11	= a high intensity of this color is present

Table 4-2 shows the position of the 2-bit color settings within the color byte. It also lists the values for the default color palette provided by the graphics library software. Each byte is listed by its number, bit settings, hexidecimal notation and color. The colors in the default palette are composed of combinations of high-intensity red, green, and blue.

There are three color procedures:

o Load Color

o LoadColorMapper

o SetColorMapper

Table 4-2. The Color Palette

Color Byte Bit Positions

8	4	2	1	8	4	2	1
		RED		GREEN		BLUE	

Byte	**Bits**	**Hex**	**Color**
1	0011 0000	30h	red
2	0011 1100	3Ch	yellow
3	0000 1100	0Ch	green
4	0000 0011	03h	blue
5	0000 1111	0Fh	cyan
6	0011 0011	33h	magenta
7	0011 1111	3Fh	white
8	0000 0000	00h	black

LoadColor

Description

Loadcolor is used to change one color in a previously defined color palette. The position of the color byte within the palette and the new value are specified, as well as the color mapper where the modified palette is to be loaded.

Procedural Interface

LoadColor (iMapper, iColor, bColor): ErcType

where

iMapper	specifies the color mapper that is to be loaded. 0 = first mapper 1 = second mapper
iColor	specifies the position within the 8-byte color palette of the color byte that is being modified. 1-8 are the valid values.
bColor	specifies the byte setting for the new color.

LoadColorMapper

Description

LoadColorMapper specifies the address of the 8-byte memory workarea used by the color mapper to create the color palette. The parameter iMapper specifies which of the two color mappers is to be loaded with the 8 bytes. The eight bytes must be formatted with the color specifications prior to calling LoadColorMapper.

Procedural Interface

LoadColorMapper (iMapper, pColors): ErcType

where

iMapper	specifies which of the two color mappers is to be loaded. 0 = first mapper 1 = second mapper
pColors	points to the memory address of the 8-byte color palette.

SetColorMapper

Description

SetColorMapper specifies which of the two color mappers is to be current. The graphics library initialization procedure sets the first mapper as the current one.

SetColorMapper can be used to switch to the other color mapper to use another color palette for the current picture.

Procedural Interface

SetColorMapper (iMapper): ErcType

where

iMapper	specifies which color mapper is to be current. 0 = first mapper 1 = second mapper

ALPHANUMERIC ATTRIBUTE PROCEDURES

The alphanumeric attribute procedures are available only on the B 21 and B 26 Color Graphics workstations. Character attributes such as blinking, half-bright, reverse video, and underlining are ordinarily under hardware control through the alphanumeric style RAM. The B 21 graphics control board and the B 26 graphics controller module each have an alternate style RAM that enables eight different attribute combinations to be used on a screen. The graphics style RAM includes color and intensity specification with reverse video and underlining. Blinking cannot be specified with this style RAM. An 8-byte memory workarea is allocated to specify the entries that are passed to the graphics style RAM. Each byte uses the low-order six bits for the color specification and the two high-order bits for reverse video and underlining respectively. Refer to the "Video Access Method" and "Video Display Management" subsections of the B 20 Operating System Reference Manual for information about using the graphics control board style RAM to access the video display.

There are three alphanumeric attribute procedures:

- o LoadColorStyleRam
- o SetStyleRamEntry
- o SetStyleRam

LoadColorStyleRam

Description

LoadColorStyleRam specifies the eight bytes that are passed to the color graphics style RAM. These attribute settings are used to display different combinations of color, reverse video, and underlining on different sections of the video screen. The low-order six bits of each byte specify the color and intensity, and the high-order two bits are used for reverse-video and underlining respectively.

Procedural Interface

LoadColorStyleRam (pAttrs): ErcType

where

pAttrs	points to the memory address of the 8-byte set of attributes.

SetStyleRamEntry

Description

SetStyleRamEntry is used to modify a single 1-byte entry in the graphics style RAM. The position of the 1-byte entry within the set of attributes must be specified as well as the value for the byte that will be modified. This function is identical to the 3.0 function LoadStyleRamEntry. LoadStyleRamEntry can still be called with 4.0 graphics; however, it will take slightly longer to execute than SetStyleRamEntry.

Procedural Interface

SetStyleRamEntry (iEntry, bVal): ErcType

where

iEntry	specifies which one of the eight bytes is to be replaced. 0-7 are the valid values.
bVal	specifies the new value for the selected entry.

SetStyleRam

Description

SetStyleRam sets a flag that indicates which style RAM is to be used; the graphics style RAM or the standard alphanumeric style RAM. The default is the standard alphanumeric style RAM.

Procedural Interface

SetStyleRam (fGraphics): ErcType

where

fGraphics	indicates which style RAM is to be used.
	0FFh = TRUE, use the graphics style RAM
	00h = FALSE, use the alphanumeric style RAM

SECTION 5
ACCESSING BUSINESS GRAPHICS

OVERVIEW

Application systems that generate tabular data can use Business Graphics to present the data graphically. Business Graphics plots three different types of graphic representations: bar charts, pie charts, and line charts. Before an application can access Business Graphics to draw a chart, two processes must be completed. All of the information needed to create the chart must be provided in parameters that are passed to Business Graphics. In addition, part of this information, the data values used by Business Graphics to plot the chart, must be arranged in a specific format.

The parameters that pass the required information to Business Graphics are set by using standard BTOS parameter procedures. A variable length parameter block (VLPB) is created in long-lived memory. The following procedures are used:

- o RgParamInit
- o RgParamSetEltNext
- o RgParamSetListStart
- o RgParamSetSimple

For detailed information about the use of these procedures, see the "Parameter Management" section in the B 20 Operating System Reference Manual. The information provided in the parameters includes the following items:

- o a picture file
- o a format file
- o a data file
- o a title
- o labels
- o a palette file

Picture File

The parameter value is actually the name of the picture file. Business Graphics opens a picture file with the specified name. After the chart is drawn, it is saved in this picture file for future use. The picture file is opened in mode write. If a picture file already exists with this name, it is overwritten.

Format File

The parameter value is the name of the format file. The format file is a skeleton. It contains the format of the chart without the data. A format file for each of the three types of charts is supplied as part of the Business Graphics software. Format files are used so that new charts can be created without extensive editing.

Data File

The parameter value is the name of the data file. The data file provides the absolute values used to draw the chart. These values are entered as standard short real numbers. FORTRAN and Pascal application programs can enter these values directly into the data file. In BASIC applications, however, the values must be converted to standard short reals. The procedure used to convert the values is:

```
     rNew = ConvertTo8087 (rOld)
```

The data file format varies depending on the type of chart that is to be drawn. Each format includes both an entry to identify the type of chart and a reserved area. After this common header, the data files differ. Detailed information about the requirements for each type of data file is found below in the descriptions of the bar chart, pie chart, and line chart procedures.

Title

The title is the label that appears above the chart. If the title parameter is not set, a default title is taken from the format file.

Labels

The type and number of label parameters vary depending on the type of chart that is to be drawn. The pie chart, for example, has labels only for its segments, while the bar and line charts can have labels for the axes and legends.

Palette File

The parameters value is the name of the color palette. If the palette file parameter is not set, the default color palette is used.

Once the data has been placed in the data file, and the parameter block has been set up, the application program can access Business Graphics. Detailed information on setting up a run file and chaining from the current application can be found in the "Task Management" section of the B 20 Operating System Reference Manual.

DRAWING A BAR CHART

Figure 5-1 is an annotated illustration of a bar chart drawn by Business Graphics.

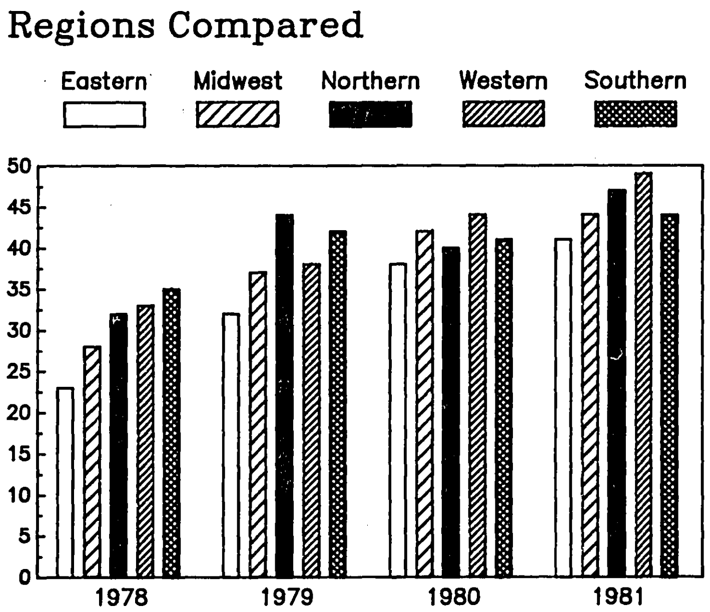


Figure 5-1. Business Graphics Bar Chart

Data File Format

The format for the bar chart data file includes an array for all of the values that are plotted for each of the legends. The complete format for the bar chart data file is described in Table 5-1.

Table 5-1. Bar Chart Data File

Item	Size	Contents
ChartType	word	0
reserved	16 bytes	0FFh
nValuesPer Legend	word	the number of values in each legend on the chart (maximum of 12)
nLegends	word	the number of legends on the chart (maximum of 5)
rgValues**	4 bytes	the absolute value for the data item, specified as a short real number

**The number of rgValues entries equals nValuesPerLegend multiplied by nLegends. The rgValues entries must be placed in the data file in the following manner:

All of the values for the first data group followed by all of the values for the next data group until all of the data groups are completed.

Parameters

The parameters for building the parameter block when the chart is a bar chart are:

BGP	
picture file	the name of the picture file that is to be used by Business Graphics to save the chart, use RgParamSetSimple (parameter = 1)

format file	the name of the format file that is to be used by Business Graphics to build the chart, the standard name is [sys]<sys>bar.fm for the B 22 or [sys]<sys>ColorBar.fm for the B 21 series, use the RgParamSetSimple (parameter = 2)
data file	the name of the data file containing the values that are to be plotted, use RgParamSetSimple (parameter = 3)
title	the label that appears above the chart, use RgParamSetSimple (parameter = 4)
X axis label	the label that appears below the X axis, use RgParamSetSimple (parameter = 5)
Y axis label	the label that appears above the chart, left-justified to the Y axis use RgParamSetSimple (parameter = 6)
legend labels	the labels that appear above the chart to describe the legends, use RgParamSetListStart (parameter = 7) and then RgParamSetEltNext for each label
X axis group labels	the labels that appear below each cluster of bars to describe the bar groups, use RgParamSetListStart followed by RgParamSetEltNext (parameter = 8)
override format	a flag that indicates whether the title and labels in the format file should be overridden, the default is "YES", enter "NO" to prevent overriding, user RgParamSetSimple (parameter = 9)

palette	the name of the color palette, user RgParamSetSimple (parameter = 10)

The title and label parameters are optional. For any that are not entered, labels from the format file are used instead. The palette file parameter is also optional. If a color palette is not specified, the default is used.

DRAWING A PIE CHART

Figure 5-2 is an annotated illustration of a pie chart drawn by Business Graphics.

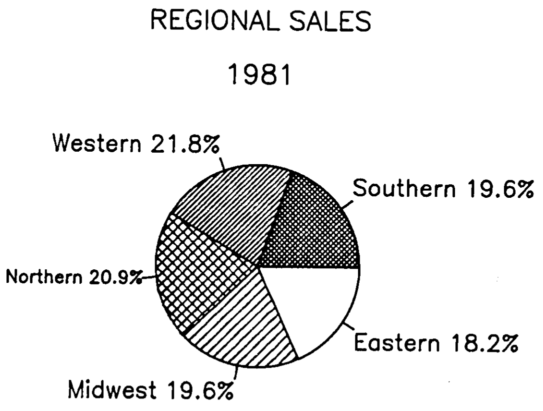


Figure 5-2. Business Graphics Pie Chart

Data File Format

The format for the pie chart data file includes an array for all of the values that are used to create segments. Using the absolute values, Business Graphics calculates the relative percentages to divide the whole circle into proportional segments. The complete format for a pie chart data file is described in Table 5-2.

Table 5-2. Pie Chart Data File

Item	Size	Contents
ChartType	word	1
reserved	16 bytes	0FFh
nSegments	word	the number of segments in the pie chart (maximum of 8)
rgValues**	4 bytes	the absolute value for a segment, specified as a short real number

** The number of rgValues entries equals nSegments.

Parameters

The parameters for building the VLPB when the chart is a pie chart are:

picture file	the name of the picture file that is to be used by Business Graphics to save the chart, use RgParamSetSimple (parameter = 1)
format file	the name of the format file that is to be used by Business Graphics to build the chart, the standard name is [sys]<sys>pie.fm for the B 22 and [sys]<sys>ColorPie.fm for the B 21 series, use RgParamSetSimple (parameter = 2)
data file	the name of the data file containing the values that are to be plotted, use RgParamSetSimple (parameter = 3)
title	the label that appears above the chart, use RgParamSetSimple (parameter = 4)
segment labels	the labels that describe the segments on the chart, use RgParamSetListStart (parameter = 5) and then RgParamSetEltNext for each label
override format	a flag that indicates whether or not the title and labels are to override the format file, the default is "YES", enter "NO" to prevent overriding use RgParamSetSimple (parameter = 9)
palette file	the name of the color palette, use RgParamSetSimple (parameter = 10)

The title and label parameters are optional. If either one is not entered, a label from the format file is used instead. The palette file parameter is also optional. If it is not entered, the default color palette is used.

DRAWING A LINE CHART

Figure 5-3 is an annotated illustration of a line chart drawn by Business Graphics. There are two different line chart formats. A line chart can be drawn with numeric values on the X axis, or alphanumeric strings can be used instead. The second type of line chart is particularly useful for describing yearly trends by months. The data file formats for both types of line charts are included below.

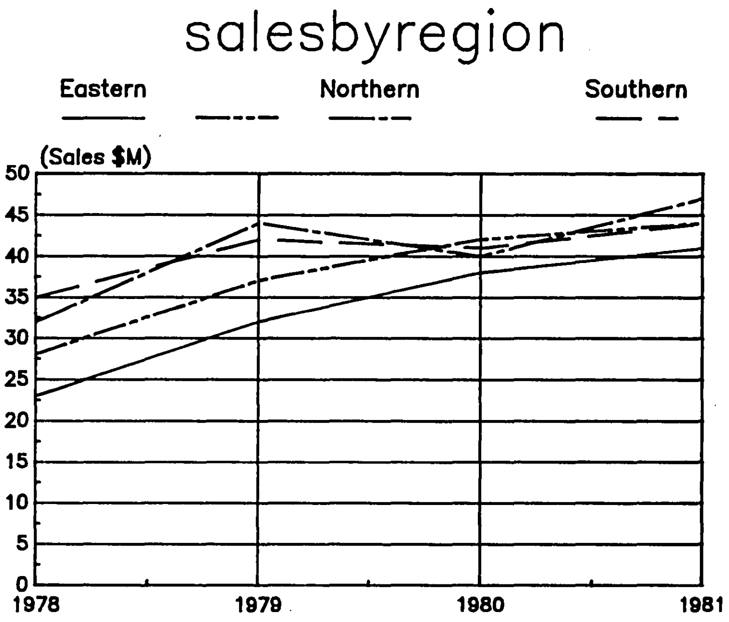


Figure 5-3. Business Graphics Line Chart

Data File Format - Numeric Line Chart

The format for the line chart data file includes an array for all of the values that are used to plot each line on the chart. The complete format for a numeric line chart data file is described in Table 5-3.

Table 5-3. Line Chart Data File

Item	Size	Contents
ChartType	word	2
reserved	16 bytes	0FFh
nLegends	word	the number of legends (and lines) on the chart (maximum of 5)
nCoordinates	word	the number of X,Y pairs for the legend (and line)
xValues**	word	the X values for the legend (and line), specified as short real numbers
yValues**	word	the Y values for the legend (and line), specified as short real numbers

** The number of xValues and yValues entries equals nCoordinates, and the number of sets of nCoordinates, xValues, yValues entries equals nLegends.

Data File Format - Alphanumeric Line Chart

Table 5-4 describes the data file format for alphanumeric line charts.

Table 5-4. Alphanumeric Line Chart Data File

Item	Size	Contents
ChartType	word	3
reserved	16 bytes	0FFh
nLegends	word	the number of legends (and lines) on the chart (maximum of 5)
nCoordinates	word	the number of X,Y pairs for the legend (and line)
yValues**	nCoordinates*4	the Y values for the legend (and line), specified as short real numbers
xValues**	variable	the X values specified as alphanumeric strings

** Each legend must have the same number of yValue entries. Because the number of Y values is fixed, nCoordinates does not have to be repeated for each legend. After all the Y values are specified, a single set of X values is entered. Each X string within the set is entered according to the following format:

- a word containing the length, in bytes, of the string
- the actual string

o   if the string length is not even, a trailer byte set to zero to give the entry an even length.

The string "JANUARY", for example, would be 10 bytes long and would be entered as follows:

```
  0,7,JANUARY,0
```

The strings are entered sequentially with no spaces or separators between each string.

Parameters

The parameters for building the VLPB when the chart is a line chart are:

picture file	the name of the picture file that is to be used by Business Graphics to save the chart, use RgParamSetSimple (parameter = 1)
format file	the name of the format file that is to be used by Business Graphics to build the chart, the standard name is [sys]<sys>line.fm for the B 22, and [sys]<sys>ColorLine.fm for the B 21 series, use RgParamSetSimple (parameter = 2)
data file	the name of the data file that contains the values that are to be plotted, use RgParamSetSimple (parameter = 3)
title	the label that appears above the chart, use RgParamSetSimple (parameter = 4)
X axis label	the label that appears below the X axis, use RgParamSetSimple (parameter = 5)
Y axis label	the label that appears above the chart, left-justified to the Y axis, use RgParamSetSimple (parameter = 6)
legend labels	the labels that appear above the chart to describe the legends, use RgParamSetListStart (parameter = 7) and then RgParamSetEltNext for each label
override format	a flag that indicates whether or not the title and labels are to override the labels in the format file, the default is "YES", enter "NO" to prevent overriding use RgParamSetSimple (parameter = 9)
palette file	the name of the color palette, use RgParamSetSimple (parameter = 10)

The title and label parameters are optional. If either one is not entered, a label from the format file is used instead. The palette file parameter is also optional. If this parameter is not entered, the default color palette is used.

AN ALTERNATIVE TO THE DATA FILE

Instead of creating a data file, an application program can allocate a portion of long-lived memory to store the data values. The data file parameter in the VLPB is set up differently to indicate that the data values are in memory. Instead of the data file name, this parameter holds a pointer to the memory address and the size of the area used. The format for these two parameters is as follows:

o a string beginning with the character "@" followed by a 4-byte pointer to the memory area.

  The low-order two bytes are the relative address, or offset.

  The high-order two bytes are the segment address.

o a binary word specifying the size of the data area.

RgParamSetSimple is ordinarily used to set this parameter. In this case, however, RgParamSetListStart must be called, followed by two calls to RgParamSetEltNext.

A SAMPLE BASIC PROGRAM

The following BASIC compiler program is a simple routine that accesses Business Graphics to create a pie chart. The input is entered through prompts on the display screen. To run this program, a version of BASIC must be configured to support calls to non-BASIC procedures. Use BasGen.Asm to create a look-up table that is used by Basic to access the non-BASIC procedures. Then, create a run file that contains the BASIC interpreter, the look-up table, and the non-BASIC procedures (including Graphics.Lib) all in object module format. For detailed information, refer to "Appendix B: Calling Non-Basic Procedures" in the Basic Compiler Reference Manual.

```
10 DIM SD%[2]
20 PMEMORY! = 0
30 NSEGMENTS% = 0
40 PIEVAL! = 0
50 ERC% = 0
60 I% = 0
70 W% = 0
80 ON ERROR GOTO 550
90 REM
100 REM create variable length parameter block
110 ERC% = ALLOCMEMORYLL(2048,PTR(PMEMORY!)): IF ERC%<>0 THEN
ERROR 99
120 ERC% = RGPARAMINIT(PMEMORY!,2048,7): IF ERC%<>0 THEN ERROR 99
130 REM
140 REM store parameters
150 A$ = "BASICDATA": I%=1: GOSUB 450
160 A$ = "[SYS]<SYS>PIE.FM": I%=2: GOSUB 450
170 A$ = "BASICDATA": I%=3: GOSUB 450
180 INPUT "TITLE OF PIE CHART ";A$: I%=4: GOSUB 450

190 REM
200 REM create data file header
210 OPEN "O", #1,"BASICDATA"
220 WIDTH 255
230 PRINT #1, MKI$(1);
240 FOR I% = 0 TO 7
250 PRINT #1, MKI$(&HFFFF);
260 NEXT I%
270 REM
280 REM get data values and store in data file
290 INPUT "NUMBER OF PIE SEGMENTS";NSEGMENTS%
292 IF NSEGMENTS%<1 OR NSEGMENTS%>8 THEN PRINT "PLEASE ENTER
NUMBER BETWEEN 1 AND 8": GOTO 290
300 PRINT #1, MKI$(NSEGMENTS%);
310 ERC% = RGPARAMSETLISTSTART(5): IF ERC%<>0 THEN ERROR 99
320 FOR I% = 0 TO (NSEGMENTS% - 1)
330 PRINT "VALUE FOR SEGMENT ";I%+1;
340 INPUT PIEVAL!
350 PIEVAL! = CONVERTTO8087(PIEVAL!)
360 PRINT #1, MKS$(PIEVAL!);
370 PRINT "LEGENDS FOR SEGMENT ";I%+1;
380 INPUT A$ : GOSUB 490
390 ERC% = RGPARAMSETELTNEXT(PTR(SD%[0])): IF ERC%<>0 THEN
ERROR 99
400 NEXT I%
410 CLOSE #1
420 ERC% = OSCHAIN("[SYS]<SYS>BGP.RUN","",129,0,0): IF ERC%<>0
THEN ERROR 99
430 END
440 REM
450 REM - store A$ as parameter I%
460 GOSUB 490
470 ERC% = RGPARAMSETSIMPLE(I%,PTR(SD%[0])): IF ERC%<>0 THEN
ERROR 99
```

```
480 RETURN
490 REM
500 REM create a pointer and length for A$
510 SD%[0] = GETRA(PTR(A$))
520 SD%[1] = GETSA(PTR(A$))
530 SD%[2] = LEN(A$)
540 RETURN
550 REM
560 REM error handler
570 IF ERR=99 THEN PRINT "ERC ";ERC%;: GOTO 590
580 PRINT "ERROR ";ERR;
590 PRINT " IN LINE ";ERL
600 STOP
610 END
```

APPENDIX A
STATUS CODES

Decimal Value	Meaning
7600	Graphics is not available on this workstation.
7601	A graphics operation was invoked without first initializing graphics.
7602	An error occurred internal to the graphics library.
7610	A graphics operation that requires a picture to be open was invoked before opening a picture.
7611	An attempt was made to open a picture when one was already open.
7612	An attempt was made to modify (vectors or labels) a picture that was opened in read mode.
7613	An OpenPicture operation failed because the picture specified does not have a valid picture name.
7614	An OpenPicture failed because the picture specified was not a picture file.
7620	A graphics operation that requires an open object was invoked when there was no open object.
7621	An attempt was made to open a picture or an object when an object was still open.
7622	The object specified to OpenObject was more than 12 characters.

Decimal Value	Meaning
7623	A graphics operation that requires an object to be closed was invoked when an object was still open.
7624	CloseTempObject was performed when the open object was not temporary.
7625	A graphics operation that requires a retained object was invoked when the open object was temporary.
7626	An attempt was made to modify (vectors and labels) the open object when the picture was opened in read mode.
7627	An attempt was made to modify (vectors and labels) the open object when the picture was opened in read mode.
7628	SetNextObject was performed when the current object was the last object in the picture.
7629	An OpenTempObject operation failed because either a picture or another object was open.
7640	A label operation requiring an open label was invoked when there was no label.
7641	SetNextLabel was performed when the current label was the last label in the object.
7642	A font name was specified that was longer than 12 characters.
7643	An invalid label origin was specified in a label operation.

Decimal Value	Meaning
7644	A text or label operation was performed in which the text string extended outside the world coordinate system.
7645	A zero-length label was specified.
7646	A graphics operation was invoked with incorrect parameters.
7647	A graphics transformation operation (SetScale, SetTranslate) would, if invoked, result in invalid transformation values.
7649	A graphics operation that was performed did not provide a large enough workarea.
7690	A printer was specified incorrectly.
7691	A font was specified incorrectly.
7692	The standard font, SimplexRoman, was not specified in the Graphics.fonts file.
7693	An invalid output device was specified.

APPENDIX B
PLOTTERS AND PRINTERS

SUPPORTED PERIPHERALS

The Graphics Support Package contains software routines that drive the following hardware peripherals:

* Burroughs AP1351 Multi Function Printer
* Burroughs B9253 Dot Matrix Printer

Burroughs Corporation supports these printers.

UNSUPPORTED PERIPHERALS

The Graphics Support Package also contains software routines which drive the following hardware peripherals:

Hewlett-Packard Model HP7220C 8 pen plotter

Hewlett-Packard Model HP7220T 8 pen plotter

Hewlett-Packard Model HP7470A 2 pen plotter

Hewlett-Packard Model HP7475A 6 pen plotter

Strobe Model 100 1 pen plotter

Printronix MVP dot matrix printer

Envision 420 dot matrix printer

Anadex 9620 dot matrix printer

Okidata Microline 93 dot matrix printer

Dataproducts 8010 dot matrix printer

None of these peripherals is marketed by Burroughs Corporation. Burroughs does not warrant the suitability or performance of these peripherals in customer applications.

The particular device selected is the responsibility of the customer.

PERIPHERAL CONFIGURATION

There are five configuration files found in the Graphics Support Package. Four of these configuration files are used to enable the unsupported plotters to interface with the graphics package. Each configuration file enables either direct or spooled plotting for either the Hewlett-Packard plotters or for the Strobe plotter. The files are:

A)	PlotterConfig.Sys	Direct for Hewlett-Packard
B)	HPPlotterConfig.Sys	Spooled for Hewlett-Packard
C)	StrobeConfig.Sys	Direct for Strobe
D)	StrobePlotterConfig.Sys	Spooled for Strobe

The printers use the following file for direct and spooled printing.

E)	GraphicsPrinterConfig.Sys	Direct and Spooled Printing

The configuration file information is needed to set peripheral switch settings. It is as follows:

Configuration files	A)	B)	C)
Data bits	7	7	8
Parity	0	0	none
Baud rate	2400	2400	2400
Stop bits	1	1	1
Transmit time out	60	60	5
Receive time out	60	60	5
CR/LF mapping mode	binary		binary
Newline mapping mode	binary	binary	binary
Line control	XonXoff	XonXoff	XonXoff
EOF byte	04		04
Tab expansion size		8	
Number of characters per line		80	
Translation file		none	

Note: If your plotter is connected to an XE520, the following changes must be made to the configure files:

* set the data bits to 8
* change the parity to none

Configuration files	D)	E)
Data bits	8	
Parity	none	
Baud rate	2400	
Stop bits	1	
Transmit time out	60	60
Newline mapping mode	binary	binary
Line control	XonXoff	
Tab expansion size	8	0
Number of characters per line	80	132
Additional ACK delay		0
Translation file	none	none

Blank sections do not apply to that type of configuration file.

For each peripheral used an entry must be made in the Sys.Printers file, and for spooled printing in the Queue.Index file. The Sys.printers entries are of the format:

```
   name 1:  spec 1 [, spec 1a]:  text type:  graphics type
   name 2:  spec 2 [, spec 2a]:  text type:  graphics type
```

Name field is a nickname or alias for the printer. This can be any character or string such as Spool or PrinterA, etc. This field is used by an application program when specifying a printer or plotter.

The specification is a device specification. If the device is to be connected directly (rather than spooled locally or through a master workstation) the specification is a configuration file (i.e., [COMM]B&[Sys]<Sys>PlotterConfig.sys). If the device is a spooled device, then the specification will be a queue name such as [SPL]. Either or both kinds of specifications may be defined for each printer or plotter. The application program determines whether the output device is installed as a spooled device (and if so, spool to it) or other (and if so, print or plot directly to it).

Text type is used to determine the type of text printer; that is, to differentiate a printer used to produce draft copy (such as a parallel) from one used to produce real formatted output (such as serial). For a plotter that has no text capabilities, this field must be blank to indicate that this device is unsuitable for printing.

Graphics type is used to determine the type of graphics formatting that must be sent to the output device. For a serial printer this field is blank.

The possible entries for graphics type are as follows:

Prism for the AP 1351 MultiFunction Printer and for the Dataproducts 8050 dot matrix printer
Oki for the Okidata Microline 93 dot matrix printer
Anadex for the Anadex 9620 dot matrix printer
Printronix for the Printronix MVP dot matrix printer
8010 for the Dataproducts 8010 dot matrix printer
HP7470A for the HP7470A 2 pen plotter
HP7220C for the HP7220C 8 pen plotter
Strobe for the Strobe Model 100 1 pen plotter
HP7475A for the HP7475A 6 pen plotter
HP7220T for the HP7220T 8 pen plotter

For example, if you are using the HP7470A plotter and are connecting directly to Channel B of your system witout using the spooler, the plotter information must be added into the Sys.printers file. The following example illustrates a customized Sys.Printers file.

```
DIRECTSER:  [PTR]B&[sys]<sys>PtrBConfig.sys:      Diablo630
DIRECTPAR:  [Lpt]&[sys]<sys>LptConfig.sys:        Draft
SERIAL:     [SplB]:                               Diablo630
PARALLEL:   [Spl]:                                Draft
DSer:       [Ptr]B&[sys]<sys>PtrConfig.sys:       Diablo630
Direct:     [Lpt]&[sys]<sys>LptConfig.sys:        Draft
DPar:       [Lpt]&[sys]<sys>LptConfig.sys:        Draft
Ser:        [SplB]:                               Diablo630
Par:        [Spl]:                                Draft
HP7470A:    [Comm]B&[sys]<sys>PlotterConfig.sys:  :HP7470A
```

This adds an entry for a plotter whose nickname is HP7470A, uses the configuration file PlotterConfig.sys, is connected through Channel B, and is of type HP7470A.

See the B 20 Software Operation Guide for more information.

SPOOLED PERIPHERAL SUPPORT

If you want to use the same environment as described above but with spooling, the following entry must be added to Sys.printers instead of the one described above:

```
HP7470A:  [COMM]B&[Sys]<Sys>PlotterConfig.sys,[HP7470A]: :HP7470A
```

This entry is the same as the previous one except that the queue name HP7470A has been added. In this case, when specifying HP7470A, it sees if the spooler is installed. If it is not, the output is sent directly through Channel B using [Sys]<Sys>PlotterConfig.sys. If the spooler is installed and that queue is installed, the output is sent directly through Channel B using [Sys]<Sys>PlotterConfig.sys. If the spooler is used for spooling to plotters, then the spooler must be installed with a spooler configuration file other than the default. This configuration file must specify the queue name to be used for the plotter, its matching printer configuration file, and its channel. In addition, the queue.index file must be modified to include this new queue, and the system must be rebooted.

The spooler configuration file which must be specified for the above example should have an entry for Channel B as follows:

```
B/HP7470A/HP7470A[Sys]<Sys>HPPlotterConfig.sys/64/n
```

where: B specifies the channel;
HP7470A is the printer name;
HP7470A is the queue name; and
[Sys]<Sys>HPPlotterConfig.sys is the printer configuration file.'

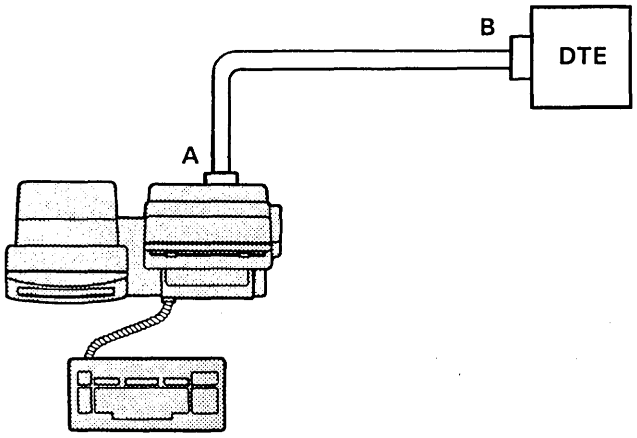
The following lines would also be added to the queue.index file:

```
HP7470A/[Sys]<Sys>HP7470A.Queue/1/1
HP7470AControl/[Sys]<Sys>HP7470AControl.Queue/1/1
```

When spooling to printers the standard parallel port line printer configuration and spooler file are to be used.

CONNECTIONS

All five of the referenced plotters require a crossed cable for RS-232-C communications. Figure B-1 illustrates the cross cable assembly.



Pinouts:

Assignments	A Workstation	B DTE	Assignments
Protective Ground (shield)	1	1	Protective Ground (shield)
Transmit Data	2	2	Transmit Data
Receive Data	3	3	Receive Data
Request to Send	4	4	Request to Send
Clear to Send	5	5	Clear to Send
Signal Ground	7	7	Signal Ground
Data Set Ready	6	6	Data Set Ready
Carrier Detect	8	8	Carrier Detect
Data Terminal Ready	20	20	Data Terminal Ready

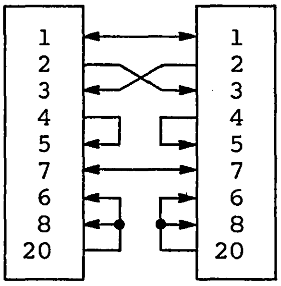


Figure B-1. Crossed Cable for RS-232-C Communications

APPENDIX C
SAMPLE GRAPHICS APPLICATION PROGRAM

This sample Pascal program opens a picture, adds an object, and draws a frame, a pattern, and a label. It also scales the object, moves it on the screen, zooms in on part of it, and changes the window to zoom in on the label.

When the program is running, depress the space bar after each picture is displayed to continue with the next picture.

```
PROGRAM Example (INPUT,OUTPUT);

CONST
        PW = ' ';
        cbPW = 0;
        ModeRead = RETYPE (WORD,'mr');
        ModeWrite = RETYPE (WORD,'mw');
        ModeModify = RETYPE (WORD,'mm');
        sMemory = #A000;
        sMemoryG = #A00;
        cbLabelEntry = 106;

TYPE
        ErcType = WORD;
        POINTER = ADS OF WORD;
        QUAD = ADS OF WORD;
        BUFFER = ARRAY [0..129] OF BYTE;
        NameType = LSTRING (255);

FUNCTION AllocMemorySL (cBytes: WORD; ppSegmentRet:
        POINTER):ErcType;
        EXTERN;

FUNCTION ErrorExit (erc : WORD):ErcType;
        EXTERN;

FUNCTION ReadKbd (pCharRet: POINTER):ErcType;
        EXTERN;

FUNCTION SetScreenVidAttr (iAttr: WORD; fOn: BOOLEAN):ErcType;
        EXTERN;

(* ________ GRAPHICS EXTERNALS ________ *)

(* ___ Initialization ___ *)

FUNCTION InitGraphics:ErcType;
        EXTERN;

FUNCTION ClearViewPort:ErcType;
        EXTERN;
```

```
(* ___ Picture Operations ___ *)

FUNCTION OpenPicture (pbPictureName: POINTER; cbPictureName:
        WORD; pbPassword: POINTER; cbPassword, Mode: WORD;
        pMemory: POINTER; sMemory: WORD):ErcType; EXTERN;


FUNCTION ClosePicture (fSave: BOOLEAN):ErcType;
        EXTERN;

FUNCTION DisplayPicture (fInterrupt: BOOLEAN):ErcType;
        EXTERN;

(* ___ Object Operations ___ *)

FUNCTION AddObject (pbObjectName: POINTER; cbObjectName: WORD;
        rXMin, rYMin, rXMax, rYMax: REAL):ErcType;
        EXTERN;

FUNCTION CloseObject:ErcType;
        EXTERN;


(* ___ Drawing Operations ___ *)

FUNCTION Draw (rX, rY: REAL):ErcType;
        EXTERN;

FUNCTION Move (rX, rY: REAL):ErcType;
        EXTERN;


(* ___ Viewing OPerations ___ *)

FUNCTION SetWindow (rXMin, rYMin, rXMax, rYMax: REAL):ErcType;
        EXTERN;


(* ___ Transformation Operations ___ *)

FUNCTION SetTranslate (rXTranslate, rYTranslate: REAL):ErcType;
        EXTERN;

FUNCTION SetTranslateRelative (rXTranslateRelative,
        rYTranslateRelative: REAL):ErcType;
        EXTERN;

FUNCTION SetScale (rXScale, rYScale: REAL):ErcType;
        EXTERN;

FUNCTION SetScaleRelative (rXScaleRelative, rYScaleRelative:
        REAL):ErcType;
        EXTERN;

(* ___ Label Operations ___ *)
```

```
FUNCTION AddLabel (rX, rY: REAL; pbString: POINTER; cbString:
        WORD; rSChars: REAL; bLorg, bPen, bUserID: WORD;
        pbFontName: POINTER; cbFontName: WORD):ErcType;
        EXTERN;

PROCEDURE TestErc (erc: ErcType);
 BEGIN
        IF erc <> 0 THEN erc := ErrorExit(erc);
 END;

PROCEDURE ClearScreen;

 BEGIN
        TestErc( SetScreenVidAttr (1, FALSE));
 END;


VAR [PUBLIC]
        Erc: ErcType;
        pMemory: POINTER;
        PicName: NameType;
        pbPicName: POINTER;
        cbPicName: WORD;
        ObjName: NameType;
        pbObjName: POINTER;
        cbObjName: WORD;
        TextString: NameType;
        pbTextString: POINTER;
        cbTextString: WORD;
        FontName: NameType;
        pbFontName: POINTER;
        cbFontName: WORD;
        bChar: BYTE;

BEGIN
        ClearScreen;

(* allocate memory for the picture file *)

 TestErc(AllocMemorySL (sMemory, ADS pMemory));

 TestErc(InitGraphics);

(* Open a picture to write into *)

pbPicName := ADS PicName[1];
PicName := 'Example.Pic';
cbPicName := WRD (PicName[0]);
TestErc( OpenPicture (pbPicName,cbPicName,ADS
PW,cbPW,ModeWrite,pMemory,sMemoryG));

(* Add object and draw into it *)
(* Coordinate range will be from (0, 0) to (50, 400) *)
```

```
pbObjName := ADS ObjName[1];
ObjName := 'Example';
cbObjName := WRD (ObjName[0]);
TestErc( AddObject (pbObjName,cbObjName, 0, 0, 50,
400));

(* Draw a frame border *)

TestErc( Move (0, 0));
TestErc( Draw (50, 0));
TestErc( Draw (50, 400));
TestErc( Draw (0, 400));
TestErc( Draw (0, 0));

(* Draw a pattern in the lower left corner *)

TestErc( Move (5, 40));
TestErc( Draw (25, 40));
TestErc( Draw (25, 150));
TestErc( Draw (5, 150));
TestErc( Draw (5, 40));
TestErc( Draw (25, 150));
TestErc( Move (5, 150));
TestErc( Draw (25, 40));
TestErc( Move (5, 95));
TestErc( Draw (25, 95));

(* Draw a label in the upper right *)

pbTextString := ADS TextString[1];
TextString := 'This is a label';
cbTextString := WRD (TextString[0]);
pbFontName := ADS FontName[1];
FontName := 'SimplexRoman';
cbFontName := WRD (FontName[0]);

TestErc( AddLabel (75, 60, pbTextString, cbTextString,
1, 6, 1, 1, pbFontName, cbFontName));

(* Pause for keyboard input *)

TestErc( ReadKbd (ADS bChar));

(* Scale down the object and redraw the picture *)

TestErc( ClearViewport );
TestErc( SetScale (0.5, 0.5));
TestErc( DisplayPicture (FALSE));

(* Pause for keyboard input *)

TestErc( ReadKbd (ADS bChar));

(* Scale the X axis independently of the Y axis *)
```

```
TestErc( ClearViewport );
TestErc( SetScaleRelative (0.3, 0));
TestErc( DisplayPicture (FALSE));

(* Pause for keyboard input *)

TestErc( ReadKbd (ADS bChar));

(* Move the object up in the picture *)

TestErc( ClearViewport );
TestErc( SetTranslate (0, 35));
TestErc( DisplayPicture (FALSE));

(* Pause for keyboard input *)

TestErc( ReadKbd (ADS bChar));

(* Make the object full size, then zoom in on part of
it *)

TestErc( ClearViewport );
TestErc( SetTranslate (0, 0));
TestErc( SetScale (1, 1));
TestErc( SetWindow (0, 0, 60, 50));
TestErc( DisplayPicture (FALSE));

(* Pause for keyboard input *)

TestErc( ReadKbd (ADS bChar));

(* Move the window to zoom in on the label *)

TestErc( ClearViewport );

TestErc( SetWindow(40, 45, 80, 75));
TestErc( DisplayPicture (FALSE));

(* Pause for keyboard input *)

TestErc( ReadKbd (ADS bChar));

(* Close the object and the picture *)

TestErc( CloseObject);
TestErc( ClosePicture (FALSE));

END.
```

APPENDIX D
MINIMUM MEMORY REQUIREMENTS

This appendix describes the minimum memory requirements for pictures and objects.

PICTURE

The minimum memory required to create a picture is 4096 bytes.

OBJECT

Each object requires 256 bytes for the header.

The requirements for the vector list vary, depending on the number and complexity of the commands that are saved. Simple commands such as SetColor require two words. FillRectangle, on the other hand, needs ten words. WriteTextString requires a minimum of ten words and needs more for long text strings.

Labels require an average of 100 bytes each. The actual amount can be more or less, depending on the length of the label text.

PRINTER

When a dot matrix printer is used for the output device, a substantial amount of memory must be allocated. Opening a picture requires approximately 30,000 bytes.

APPENDIX E
GLOSSARY

Aspect Ratio. The aspect ratio is the ratio of height to width for the video display area.

Business Graphics. Business Graphics is a high-level, menu-driven graphics application system that can be accessed from user-designed applications.

Character Size. Character size is a text attribute that specifies the relative size of the characters in a text string.

Color Palette. The color palette is a set of eight colors that can be used for color specification on the B 21 series Color Graphics workstation display or on plotter output.

Device-Dependent Procedures. Device-dependent procedures are graphics library procedures that can be used exclusively in either B 22 Graphics workstation applications or B 21 series Color Graphics workstation applications.

Device-Independent Procedures. Device-independent procedures are graphics library procedures that can be used in applications designed to run on any B 20 Graphics workstation.

Drawing Attributes. Drawing attributes are variable characteristics of an object, such as line type and color, that are stored with drawing commands in the vector list of an object.

Drawing Mode. The drawing mode is a drawing attribute that describes the method by which a vector or arc is written to display memory.

Font. The font is a text attribute that specifies which font is used for the character set when a label or text string is displayed.

Graphics Library (Graphics.Lib). The graphics library is a set of system-level procedures that can be called from user-designed applications to use all the capabilities of the graphics software.

Label List. The label list is the component of an object where label text and attributes are stored.

Label Origin. The label origin is a text attribute that specifies how text is to be oriented in relation to the current display position.

Line Type. The line type is a drawing attribute that describes the pattern of dots and dashes used when a vector is drawn.

Normalized Device Coordinate System. The normalized device coordinate system is a device-independent coordinate system used to reference the video display screen. The coordinate units describe positions in terms of their relation to the top, bottom and sides of the display area.

Object. An object is a structural component of a graphic representation. It is a set of graphic commands and labels that can be edited and manipulated as an entity.

Picture. A picture is the main structural component of a graphic representation. A picture is composed of one or more objects.

Picture File. A picture file is a file that is used to save a picture so that it can be redisplayed.

Raster mode. Raster mode is a method of writing graphic representations. Patterns of bits are copied onto rectangular areas of display memory.

Temporary Object. A temporary object is an object for which no vector and label data are saved. Temporary objects can not be redisplayed.

Text Attributes. Text attributes are variable characteristics of an object, such as font and character size, that are used and stored with labels or text strings.

Transformation List. The transformation list is the structural component of an object where translation and scalar units are stored.

User-Defined Coordinate System. A user-defined coordinate system is a device-independent coordinate system where the coordinate unit ranges are specified by the user and automatically mapped to the world coordinate system by the graphics software.

User-Written Procedures. User-written procedures are device-independent routines called by the graphics software that can be replaced by user-designed versions.

Vector List. The vector list is the structural component of an object where commands, drawing attributes, and text strings are stored.

Viewport. The viewport is the portion of the video display screen that defines where a graphic representation is to be displayed.

Window. The window is the portion of the world coordinate system that defines what is to be displayed on the video screen. Coordinate positions outside the window are clipped.

World Coordinate System. The world coordinate system is a device-independent coordinate system used internally by the device-independent graphics library software to map objects to display memory.

APPENDIX F
KNOWN LIMITATIONS

1. When graphics is used on a B 26 (with release level 4.0 or earlier Standard Software) and configured with one megabyte of memory, the system hangs (it does not respond to keyboard entries).

2. To set a B 22 to 132 column mode with Graphics, the system must first have an InitGraphics done at 80 column mode. Either mode can then be set.

3. Grfx-5.0-Update.Sub must be submitted in order for 4.0 Graphics to work with 5.0 BTOS. Before submitting, copy Grfx-5.0-Update.Sub into the directory containing Graphics.Lib. This creates a 'special' release of Graphics.Lib. Use this with 5.0 CTOS.Lib to relink 4.0 graphics applications. This allows you to create runfiles that work on 5.0 BTOS.

4. The two leftmost columns of pixels do not display on a B 26. Use SetViewPort to adjust the viewing display.

5. The default descriptions for LoadPaper and SetPen (user-written procedures) are not implemented in this release. Users can still replace these procedures.

INDEX

NOTES

NOTES

Documentation Evaluation Form

Title: B 20 Systems Graphics Programmer's Guide (Release Level 4.0)

Form No: 1180098

Date: May, 1985

Burroughs Corporation is interested in receiving your comments and suggestions regarding this manual. Comments will be utilized in ensuing revisions to improve this manual.

Please check type of Comment/Suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error ☐ Other

Comments: ____

From:

Name ____

Title ____

Company ____

Address ____

____

Phone Number ____________________ Date ____________________

Remove form and mail to:

Burroughs Corporation
Corporate Product
Information East
209 W. Lancaster Ave.
Paoli, PA 19301 U.S.A.