UNISYS

BTOS Graphics

Programming Reference Manual

Relative to
Release Level 1.3

Priced Item

March 1987
Distribution Code SA
Printed in U S America
1182706

UNISYS

BTOS Graphics

Programming Reference Manual

Relative to
Release Level 1.3

Priced Item

March 1987
Distribution Code SA
Printed in U S America
1182706

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication should be forwarded, using the Product Improvement Card at the back of this manual, or remarks may be addressed directly to Unisys Corporation, Corporate Product Information East, Building C, P.O. Box 500, Blue Bell, PA 19424 U.S.A.

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

Burroughs AP 1311 Multi Function Printer

Burroughs B 9253 dot-matrix printer

Burroughs AP 1351 (80-column mode and 132-column mode) multifunction printer

Burroughs AP 1351-1 (80-column mode and 132-column mode) multifunction printer

Burroughs AP 1314 dot-matrix printer

Burroughs AP 1354 (80-column mode and 132-column mode) dot-matrix printer

Burroughs AP 9208 (portrait mode and landscape mode) non-impact laser printer

These printers are supported by Burroughs Corporation.

The Business Graphics Package also contains software routines which drive the following hardware peripherals. However, none of the following peripherals is marketed by Burroughs Corporation. Burroughs does not warrant the suitability of performance of these peripherals in customer applications.

Hewlett-Packard Model HP 7220C 8-pen plotter

Hewlett-Packard Model HP 7220T 8-pen plotter

Hewlett-Packard Model HP 7470A 2-pen plotter

Hewlett-Packard Model HP 7475A 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.

Contents

Title Page

Title **Page**

Title Page

Overview

Graphics is a software interface to Burroughs graphics hardware. With this product, you can create flexible, high-speed graphics application systems, add graphics illustrations to on-line applications, produce graphic representations of data, or develop complex graphic illustrations. Burroughs applications of BTOS graphics include BTOS Systems Draw, Business Graphics Package, and the Tektronix 4014 Emulator.

Figures 1-1, 1-2, and 1-3 illustrate graphics packages created using BTOS graphics calls. Figure 1-1 is a map created using GEO/BASEMAP, which contains calls from BTOS graphics. GEO/BASEMAP is an automated system for mapping and spatial analysis developed by GEOGROUP of Berkeley, California. Figure 1-2 , drawn using BTOS Systems Draw, illustrates a BTOS Systems Draw screen. Burroughs applications programmers used Graphics Manager to create this easy-to-use, menu-driven, general purpose drawing system. Figure 1-3 illustrates a Business Graphics Package screen. BGP, a Burroughs product created using Graphics Library calls, translates Multiplan spreadsheets into bar, pie, and line charts.

Graphics provides two levels of support:

Graphics Manager and Graphics Library. Graphics Library contains calls to the Graphics Manager, a system service.

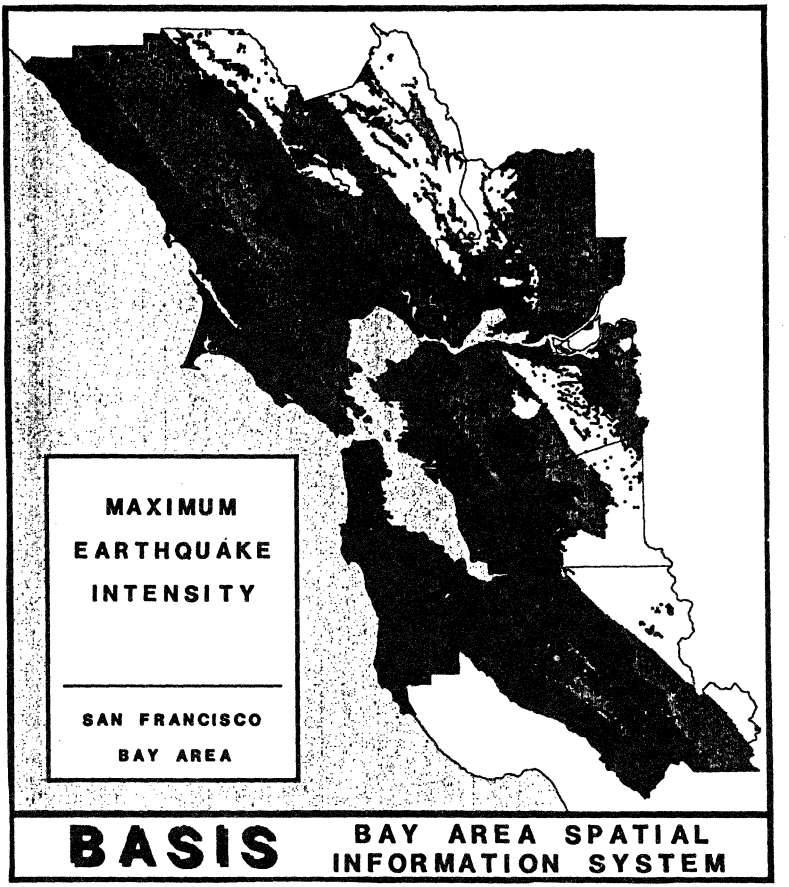
Graphics Manager

Graphics Manager is a system service that is incorporated into your operating system during installation. Each BTOS workstation has a customized Graphics Manager. Graphics Manager is a fast, integer-based set of drawing and control primitives that provides low-level graphics support. It provides a quick display of temporary images, has a low memory overhead, and allows high-speed graphics drawing. Graphics Manager supports Pascal, BASIC, COBOL, and FORTRAN. (COBOL is not supported on workstations with less than 640K of memory.)

Graphics Library

Graphics Library is a set of system-level procedures that are called from user-designed applications. It provides high-

Figure 1-1 **Map Created Using GEO/basemap (Courtesy of Paul Wilson, GEOgroup.)**



level, device-interchangeable access to the graphics hardware. Graphics Library consists of object code that is linked with your application programs. Graphics Library can be used to draw vectors and arcs, and to print and plot your graphic creations on a number of printers and plotters. You can label graphic representations with a variety of fonts, colors, and character sizes. You can use Graphics Library to create complex graphic representations by merging different figures in the same display. You can save what you have created with Graphics Library procedures on your

Figure 1-2 **BTOS Systems Draw Screen**

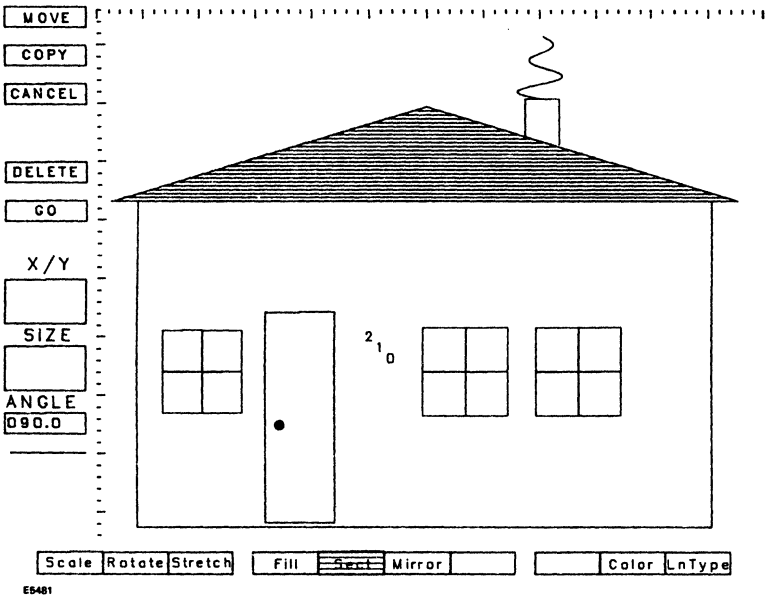
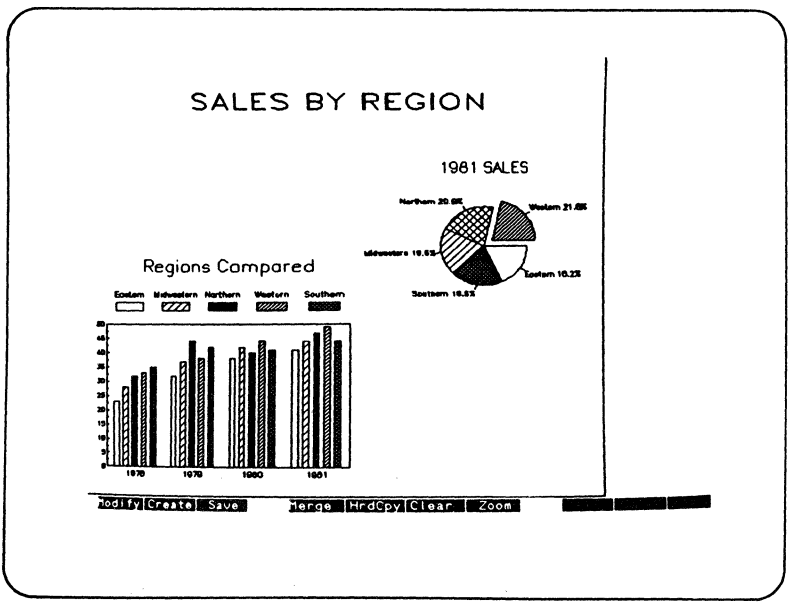


Figure 1-3 **Business Graphics Package (BGP) Screen**



floppy or hard disk for future use. You can use Pascal, BASIC, and FORTRAN with Graphics Library. (COBOL is not supported because COBOL's intermediate code file is too large to be compiled.)

Once you have created a graphic representation using Graphics Library, you can easily transform it to assume different sizes, shapes, and positions on your screen. These transformations are handled independently without altering the code that you used to create the picture. You can magnify small sections and change the size and shape of the display.

Graphics Manager and Graphics Library Features

Table 1-1 presents some of the major differences between Graphics Manager and Graphics Library. Many features are not listed; this is simply a guide you can use to determine which of the two is best suited to your particular programming needs.

Table 1-1 **Comparison of Graphics Manager and Graphics Library**

Feature	Graphics Manager	Graphics Library
COBOL	With 640K	Not Supported
Direct Access to Hardware	Yes	No
Execution Speed	Faster than Graphics Library	Slower than Graphics Manager
Interchangeable between workstations	With Adjustments	Automatically
Memory Requirements	27K minimum	160K minimum
Printers/Plotters	No	Yes
Save Graphic Design	No	Yes
Scale Drawn Figures	No	Yes
Translations	No	Yes

Where to Look Next

Read Sections 2 and 3 for information about equipment requirements and installation. Read Section 4, Concepts, for more information about Graphics Manager, Graphics Library, and the package in general. Read Section 5 if you are unfamiliar with Burroughs variable name conventions. Section 6 contains information about directly accessing graphics memory; section 7 contains detailed information about Graphics Manager procedures; section 8 contains detailed information about Graphics Library procedures. Read Section 9 for information about configuring applications for graphics. Section 10 contains information about printers and plotters. The appendixes contain sample programs, system walkthroughs for Graphics Manager and Graphics Library, status codes, tables of technical data, and a glossary of relevant terms.

Hardware and Software Requirements

Hardware Requirements

In order to run Graphics, you need one of the following systems:

- B 21 color standalone, master, or cluster with color monitor
- B 22 standalone, master, or cluster, with graphics board and monochrome monitor
- B 26 standalone, master, or cluster, with graphics module and either color or monochrome monitor
- B 27 standalone, master, or cluster with graphics module and either color or monochrome monitor
- B 28 standalone, master, or cluster, with graphics modules and either color or monochrome monitor
- XE 520 master (with BTOS workstations)

Note: *Graphics is not supported on BTOS dual floppy standalone workstations or on B 27 Low Cost Cluster workstations.*

You need at least 512 kilobytes of random access memory (512K) to run Pascal, BASIC, and FORTRAN, and 640K to run COBOL. You cannot run COBOL on a B 21 because of this memory requirement.

Graphics Workstation Features

B 21 Workstation: Burroughs workstation featuring an 80 character by 28 scan line video display screen, one cursor on the screen, a 256 character set that cannot be modified by software, and a screen split horizontally into multiple frames. The graphics resolution of the B 21 is 432 by 319 pixels. The 64-color monitor is capable of displaying eight colors simultaneously.

B 22 Workstation: Burroughs workstation featuring a 34 line screen, a software-selectable 80- or 132- character line, one cursor per line, a 256 character set that can be dynamically modified by software, and a screen that can be split horizontally and/or vertically into multiple overlapping frames. The B 22 features a high-resolution monochrome monitor with a graphics resolution of 656 by 510 pixels. To run graphics with this workstation, you need a graphics board.

B 26 Workstation: Burroughs workstation featuring a video screen display with a 29 scan line by 80 character screen, one cursor per line, a 256 character set that can be dynamically modified by software, and a screen split horizontally and/or vertically into multiple overlapping frames. The B 26 features either a monochrome monitor or a 64-color monitor that can display eight colors simultaneously, with a resolution of 718 by 348 pixels. To run graphics with this workstation, you need a graphics module.

B 27 Workstation: Burroughs workstation featuring a software-selectable vertical and horizontal character resolution of 30 (or 34) line by 80-or 132-characters, one cursor per line, a 256 character set that can be dynamically modified by software, and a screen that can be split horizontally and/or vertically into multiple overlapping frames. The B 27 features either a monochrome or color monitor capable of displaying simultaneously eight of a possible 256 colors, with a resolution of 720 (or 792) by 480 pixels. To run graphics with this workstation, you need a graphics slice.

B 28 Workstation: Burroughs workstation featuring a video screen display with a 29 scan line by 80 character screen, one cursor per line, a 256 character set that can be dynamically modified by software, and a screen split horizontally and/or vertically into multiple overlapping frames. The B 28 features either a monochrome monitor or a 64-color monitor that can display eight colors simultaneously, with a resolution of 718 by 348 pixels. To run graphics with this workstation, you need a graphics module.

Software Requirements

You will need the BTOS operating system, 7.0 or higher.

You can use any of the following BTOS 5.0 languages:

- Pascal
- BASIC Interpreter
- BASIC Compiler
- FORTRAN
- COBOL

Note: *If you want to use COBOL, you need at least 640K of memory. COBOL is not supported for Graphics Library functions. It will work with Graphics Manager on BTOS workstations with at least 640K of memory.*

Note: *To use Pascal on a B 28 workstation, first relink Pascal on the B 28.*

Installation

Before you can use Graphics, the Tektronix 4014 Emulator, Business Graphics Package, or BTOS Systems Draw, you must install Graphics and invoke the Graphics Manager.

Installing Graphics

To install Graphics on your BTOS workstation (standalone, cluster, master, or XE 520 master), use the following procedure for both 8-inch and 5¼ inch floppy disks:

1 Insert the B20GM1 floppy disk into disk drive f0.

2 Type **Software Installation**.

3 Press RETURN.

You will see these parameters:

```
[CMD FILE]
[FILES TO]
[Confirm?]
[Install file]
```

4 Enter the name of the appropriate command file after the [CMD FILE] prompt if you do not want to use the default, <Sys>Sys.Cmds, on the volume on which you are installing the software.

5 Press GO.

6 Follow the instructions on the screen until the following message appears:

```
END OF INSTALLATION OF GRAPHICS FOR BTOS SYSTEMS
```

7 Reboot your system to complete the Graphics installation.

Graphics for BTOS is now installed on your system. Remove the floppy diskette from drive F0 and place it in a safe place.

Invoking Graphics

To invoke graphics, type the following from the Executive:

1. Type **Install graphics manager** or a unique abbreviation.

2 Press GO.

Invoking Graphics Automatically

If you want your system to automatically invoke Graphics for BTOS Systems every time you logon or reboot your system, you can use a JCL file to do this. To put graphics in a JCL file, complete the following steps.

1 Type **Edit**; then press RETURN.

2 Type **SysInit.JCL**; then press GO.

3 Add one of the following lines before the $END statement:

If you used default values and installed Graphics for BTOS Systems on your master during software installation, type:

```
$run [!Sys]<Sys>InstallGM.run
```

If you installed Graphics on your local hard disk, type:

```
$run [Sys]<Sys>InstallGM.run
```

4 Press FINISH.

5 Type **Y** and press GO.

The next time you reboot your system, Graphics Manager will be installed automatically.

B20GM1 File Contents

<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>Install.sub
<Sys>MasterInstall.sub

<Fonts>ComplexRoman.font
<Fonts>DuplexRoman.font
<Fonts>SimplexRoman.font
<Fonts>SimplexPlot.font
<Fonts>Gothic.font
<Unisys>GraphicsPrinterConfig.sys
<Unisys>Graphics.lib
<Unisys>MapGraphicsWindow.obj
<Unisys>InstallGM.run

<Unisys>GM.run
<Unisys>LaserPrinterConfig.sys
<Unisys>gfxGPAMDum.obj
<Unisys>ReadVDM.lib
<Unisys>Request.Graphics.sys
<Unisys>HPPlotterConfig.sys
<Unisys>PlotterConfig.sys
<Unisys>StrobeConfig.sys
<Unisys>StrobePlotterConfig.sys
<Config>Sample>Sys.Printer
<Config>Sample>SplCnfg.Sys
<Config>Sample>Queue.Index

*These files are found only on 8-inch floppy disks and are not on 5¼ inch floppy disks.

Notes:

The file ReadVDM.lib *is for future use; its use is not currently supported.*

The file Graphics.fonts *is created during installation.*

Concepts

This section contains general information about Graphics. It contains explanations of terms and concepts that will help you understand procedures described later in this manual.

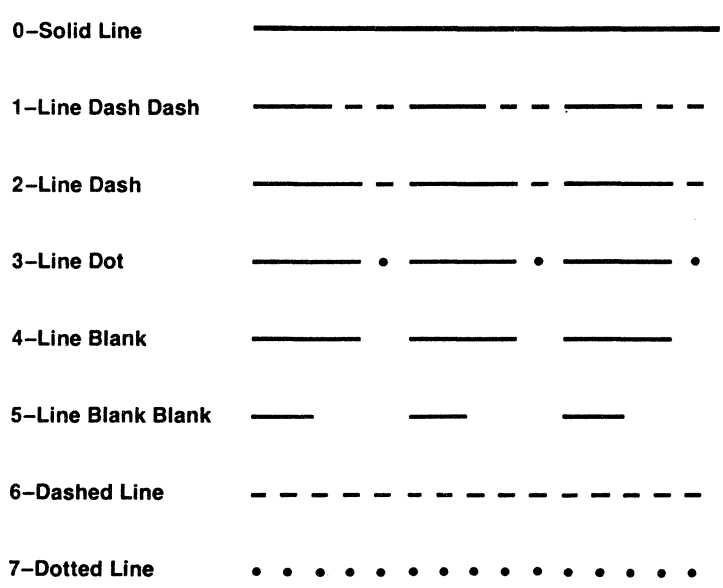
Drawing Attributes

Attributes are variable characteristics connected with vectors, arcs, and text. Drawing attributes are used with drawing procedures to provide variety and contrast in graphic representations. The attributes that are used with drawing procedures are line type, drawing mode, and color.

Line Type

Graphics includes eight system-defined line types and up to eight user-defined patterns. A solid line is the default, and there are other patterns of dots and dashes. Figure 4-1 illustrates the eight line types provided with the Graphics software. Note that only the eight system-defined line types will operate with Graphics Library software.

Figure 4-1 **Line Types**



Drawing Mode

The drawing mode describes the method used to write a vector or arc to the display screen. 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 the display memory (screen) according to the drawing mode. Four modes are supported by the library:

1 Set (OR)

2 Clear (Reset or set with matt color)

3 Complement (XOR)

4 Replace

Figure 4-2 shows examples of the same pattern written to the display memory in each of the four modes.

Set Mode

Logically ORs the pattern of the line to be drawn with the background 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.

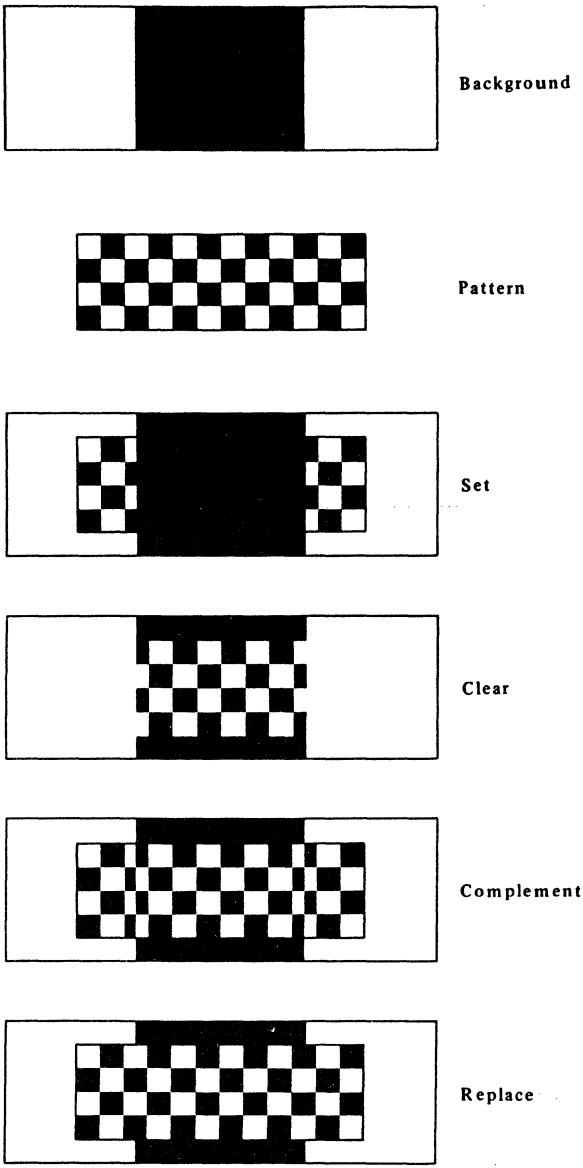
Complement Mode

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

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 4-2 **Drawing Modes**



Appendix B illustrates how line drawing is affected by line patterns, foreground, color and drawing modes.

Color

You can choose from a range of 64 colors on all BTOS color graphics workstations. The color procedures allow for the definition of 8-byte color palettes that have one byte for each of eight colors. This set of colors is called the color palette.

Two eight-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 Graphics Library procedures such as SetColor, but Graphics Manager functions handle the specification of the colors that make up a palette. The Graphics Manager procedures use the color style RAM on the B 21 graphics control board and the B 26 graphics controller module.

The color palette is automatically saved with the accompanying picture in the picture file in the Graphics Library, thus enabling pictures to be redisplayed with the same color specifications.

Color Graphics Attributes

Color alphanumeric attributes are supported on BTOS graphics workstations. Refer to *BTOS Reference Manual Volumes 1 and 2* if you need this information.

Color Printing and Plotting

You can print to a color plotter on any BTOS workstation, even those with monochrome display screens. The color parameters are used to specify the pen numbers of the plotter. With these parameters, you can select a maximum of eight colors, assigning a plotter pen number for each one.

Foreground and Background Colors

The current drawing color is considered the foreground drawing color. This color replaces pattern bits of 1 in the set and replace mode.

The background color is palette entry 0. It is used by the system during initialization and ClearScreen operations to set the color of the screen.

Mixing Graphics Library and Graphics Manager Calls

Use caution if you use both Graphics Library and Graphics Manager calls in one program. Only Graphics Library calls are saved in an object or picture file and reproduced when you issue a DisplayPicture or a DisplayCurrentObject call. If Graphics Manager calls are mixed in with Graphics Library calls, the Graphics Manager calls will *not* be saved and will not be displayed when a DisplayCurrentObject or a DisplayPicture is issued. Mixing Graphics Library and Graphics Manager calls is not recommended.

Graphics Memory Access

The Graphics Memory Access routine is available on all workstations. It enables a graphics application to directly access the graphics bitmap. This routine can be linked in with your application in order to provide it with CPU addressing to the graphics hardware.

Graphics Manager Procedures

Graphics Manager is a system service. Because Graphics Manager procedures are executed entirely by the graphics hardware, you must either modify your applications according to specific workstations or write applications that can be run interchangeably on different workstations by using virtual coordinates and scaling drawn objects.

The requests provided by the Graphics Manager fall into three functional categories: control functions, vector and arc manipulation functions, and color functions. The color procedures are available only on workstations that support color.

The routines contained in the Graphics Manager can be accessed through either the standard BTOS request interface or the procedural interfaces defined for these routines. Request block formats and their respective procedural interfaces are provided in Section 7.

See Appendix E for a Graphics Library system walkthrough.

80- or 132-Column Mode on B 27 Workstations

If you have a B 27 monochrome workstation, you can choose to have either 80- or 132-columns using the Graphics Manager. Each mode uses its own aspect ratio to calculate for a square pixel appearance. The application is able to switch between modes at any time. The picture on an 80 column mode screen will disappear and the memory will clear when you switch column modes. Appendix C contains the physical and virtual coordinate systems for both the 80- and 132-column mode.

In order to switch modes in the Graphics Manager, you need to issue requests to the video hardware. This will reset both the character screen and the graphics screen to the mode you request. Refer to *BTOS Reference Manual Volumes 1 and 2* for further details.

Note: *132-column mode is not available on B 27 color graphics workstations.*

The Graphics Manager Coordinate System

The Graphics Manager software uses both the real and virtual coordinate systems. Workstations with non-square pixels (B 26 and B 27) provide a virtual, square-pixel interface. (See Appendix C for a table of both the virtual and physical screen resolutions for both the 80- and 132-column mode of BTOS workstations.) Raster procedures, described later, use their own coordinate system.

Real Coordinate System

The real coordinate system is the horizontal and vertical number of pixels on the screen. The lower left corner of the screen is always (0,0). The B 21 and B 22 drawing procedures accept only real coordinates.

Virtual Coordinate System and Square Pixels

The virtual coordinate system is used to simulate square pixels for Graphics Manager. If you used real coordinates with non-square pixels, a horizontal line 20 pixels long would not be the same size as a vertical line 20 pixels long. Therefore, the Graphics Manager uses the virtual coordinate system, and the B 26, B 27, and B 28 drawing procedures accept virtual coordinates only. These procedures internally translate the virtual coordinates into real coordinates. When virtual coordinates are used, horizontal and vertical lines of the same length will appear to be the same length on the screen.

Control Procedures for B 22 and B 27 Workstations

Control routines are used to control the number of graphics pages and the output to the video display screen. There are two pages available on the B 22 and B 27 workstations; however, only one of them can be displayed at any given time. Only the B 22 and B 27 have multiple pages.

Vector and Arc Manipulation Procedures

Vectors and arcs are plotted by calculating lines between endpoints. B 26 and B 27 drawing commands use the virtual coordinates described in Appendix C to present the illusion of square pixels. Additional drawing commands that use real coordinates are provided. These commands also allow filling and clearing of rectangular areas.

Graphics Manager Restrictions

The level of support provided by the Graphics Manager depends on the capabilities of your workstation. Although the Graphics Manager contains requests for color procedures, they will not be supported on workstations that are not equipped for color.

Graphics Library

Graphics Library consists of object code that is linked with your application programs. The Graphics Library contains calls to the Graphics Manager and the operating system.

Graphics Library can be used to draw vectors and arcs, and to print and plot your graphic creations on a number of printers and plotters.

Appendix E contains a Graphics Library system walkthrough.

Note: *To link with the Graphics Library (Graphics.lib), the file* <Sys>gfxGPAMDum.obj *must be specified as one of the files in the* object modules *field of the linker form. This file resolves several entry points that are not defined in Graphics.lib. Failing to include* <Sys>gfxGPAMDum.obj *in the linker form would result in several "unresolved external" errors.*

80- or 132-Column Mode

The Graphics Library supports 80- or 132-column mode switching on the monochrome B 27. The picture is cleared from the screen when a change occurs in column mode. Issuing a DisplayPicture call will display the picture in the new mode. 132 column mode is a higher resolution than the 80 column mode. The resolution is the only difference in the picture on the screen.

Switching column modes will clear the character memory and create one video frame (the entire screen) for character use. There is no effect on text written with the text procedures in the Graphics Library.

User-Replaceable Procedures

Graphics Library contains three user-replaceable procedures that you can customize with your own code to expand the capabilities of the software. The existing versions of these procedures contain default values that you can replace. One is used to interrupt the process of displaying a picture; the others write messages to your screen during the operation of your plotter.

Pictures and Objects

In Graphics Library, graphic representations are called *pictures*. A picture consists of one or more objects. An object is a set of graphics commands and labels that can be edited and manipulated as an entity. Pictures and objects are the structural components of graphic representations. One bar chart on the screen, for example, is a picture with one object. A pie chart, a line chart, and a bar chart together on the screen is a picture with three objects. Objects can also overlay each other in pictures. If you want to save a graphic representation (not a temporary object) , you must open a picture before you perform any drawing or text labeling. Once a picture is open, non-temporary objects can be created. This discussion refers to non-temporary objects.

Several objects 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:

- List of vector and text commands
- List of labels (text and attributes)
- List of transformation values

Vector List

The graphic portion of the data representation is collected in a vector list that stores the commands 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. You cannot modify individual commands within the vector list, but the entire list can be cleared and rebuilt to modify the object.

Label List

Labels are notes that accompany the vector portion of the object. The label list consists of the text and the attributes of each label. The attributes include 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. Several types of modifications can be made to label text. You can change the font, change the text itself, and move the label, for example, without altering the vector portion of an object.

Transformation List

Once an object has been created, you can alter the object's size, shape, and position within the picture. These alterations translate and scale the object, and save the translation and scalar units in a *transformation list*. Although you have altered what appears on the screen, the code that created the original image is not altered by scaling and translating an object. The original object is unaffected in the label and vector lists. Therefore, when you use the DisplayPicture call, the original object will be displayed on the screen, rather than the object as it appeared after you scaled and translated it.

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 be defined as temporary. Temporary objects are used when a picture is not open. For quick graphic representations used in testing, demos, or initial system development, this definition provides 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 attributes are available.

Text Attributes

Attributes are 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 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 standard 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 five fonts: SimplexRoman, ComplexRoman, DuplexRoman, SimplexPlot, 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 used for these fonts. The default font is SimplexRoman. The names are defined in the file Graphics.Fonts.

Label Origin

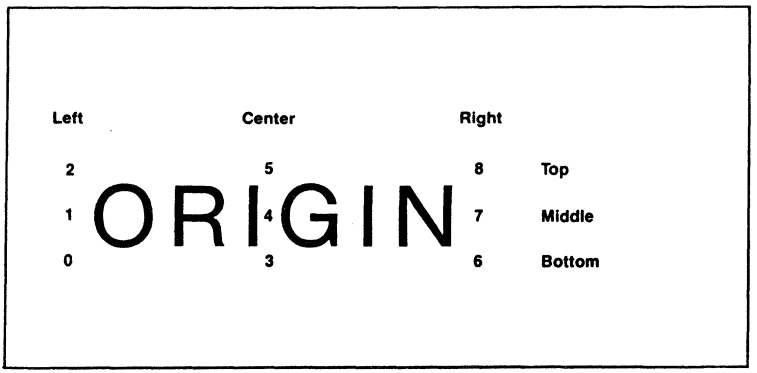
The label origin attribute indicates how text should be oriented in relation to the current display position. Text can be placed flush left, flush right, or centered at the current position, and it can begin at the top, middle, or bottom of the current position. Figure 4-3 illustrates the label origin positions.

Graphics Library Coordinate Systems

Graphics Library coordinates, instead of the video screen's physical 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

Figure 4-3 **Label Origin**



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, label, 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.

The video display screen does not represent a square area. Only the portion of the world coordinate system that represents the aspect ratio—the ratio of height to width of the physical screen—is generally used. The aspect ratio coordinate positions that represent the world coordinate values and the Normalized Device Coordinate values for BTOS workstations are shown in Appendix D.

User-Defined Coordinate System

The device-independent Graphics Library procedures also support user-defined coordinate systems. Once the user defines the minimum 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

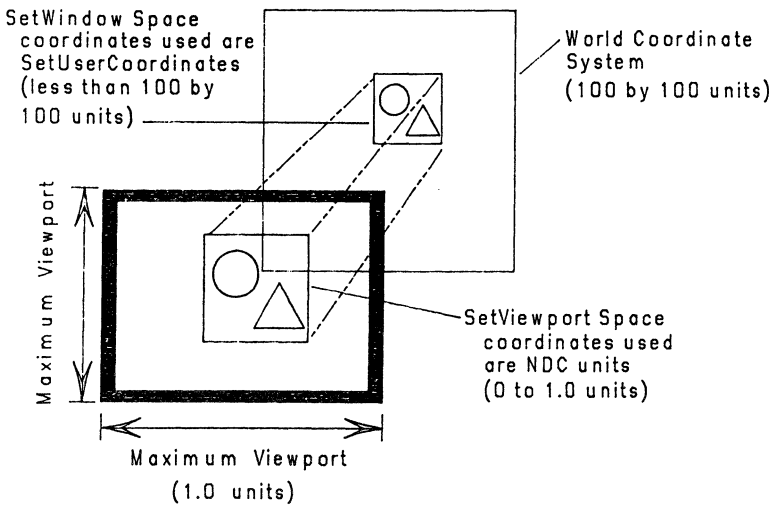
This system is used to reference the video display screen in a relative way. The coordinate positions theoretically range from (0.0, 0.0) at the lower left corner to (1.0, 1.0) at the top right. Appendix D lists the Normalized Device Coordinates for BTOS workstations. The coordinate units describe positions in terms of their relation to the top, bottom, right, and left sides of the display area. These positions refer to the video display screen, not to a picture you want to display. Currently, the only procedures that use this coordinate system are the cursor control functions and the viewport procedures. Appendix D gives the Normalized Device Coordinates for BTOS workstations.

Viewing Perspectives

The window is a portion of the world coordinate area that defines what you want to display. The viewport is a portion of the screen where you want the information in the window to be displayed. Figure 4-4 illustrates the mapping between the window and the viewport. Window/viewport transformations enable you to view pictures 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.

You can alter the perspective for viewing a picture dynamically by adjusting the window and viewport sizes, shapes, or positions. For example, you can scan a large picture by keeping the size of the window constant but changing its position within the world coordinate area. You can magnify a small section of a picture 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.

Figure 4-4 **Window to Viewport Transformation**



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 can be displayed. Ordinarily, part of the screen area is reserved for messages and forms (depending on the application task). Therefore, the viewport is usually smaller than the area of the entire screen.

A window cannot be larger than the portion of the world coordinate area that corresponds to the aspect ratio of the display screen. (See Appendix D.) You can set and reset the window to define different portions of the entire world coordinate area. To define the lower left quadrant of a B 27 display area as a window, for example, use the boundaries (0,0) to (50,33.33). You will be able to see only the coordinates outlined by the boundaries of the window, regardless of the objects that have been mapped to positions in the world coordinate area. All of the coordinate positions outside the window will be clipped.

Standards and Conventions

Numbers

Numbers are decimal except when suffixed with *h* for hexadecimal or *b* for binary. For example, 10h = 0001 0000b = 16, and 0FFh = 1111 1111b = 255.

Memory Address

Memory address refers to the logical memory address.

Variable Names

Variables are named according to their characteristics. Parameters used in procedure definitions, fields of request blocks, and other data structures are named according to this convention.

A variable name is composed of up to three parts: a prefix, a root, and a suffix.

Prefixes

The prefix identifies the data type of the variable. Common prefixes include the following:

b	Byte (8-bit character or unsigned number)
c	Count (unsigned number)
f	Flag (TRUE = 0FFh or FALSE = 0)
i	Index (unsigned integer)
n	Number (unsigned number) (same as c)
o	Offset from the segment base address (16 bits)
p	Logical memory address (pointer) (32 bits consisting of the offset and the segment base address)
q	Quad (32-bit unsigned integer)
r	Four-byte short real number, 8087 (IEEE) format.
rg	Array

s	Size in bytes (unsigned number)
sb	Array of bytes where first byte is the size
w	Word (16-bit)

Prefixes can be compounded. Common compound prefixes are:

cb	Count of bytes (the number of bytes in a string of bytes)
pb	Pointer to (logical memory address of) a string of bytes
rgb	Array of bytes

Roots

The root of a variable name can be unique to that variable, selected from the following list, or it can be a compound of the two. Common roots include:

dcb	Device Control Block
erc	Status (error) Code
exch	Exchange
fcb	File Control Block
fh	File Handle
lfa	Logical File Address
ph	Partition Handle
qeh	Queue Entry Handle

rq	Request Block
ucb	User Control Block

Suffixes

The suffix identifies the use of the variable. Suffixes are:

Last	The largest allowable index of an array.
Max	The maximum length of an array or buffer (thus one greater than the largest allowable index).
Ret	Identifies a variable whose value is to be set by the called process or procedure rather than specified by the calling process.

Examples of Variable Names

cbFileSpec	The count of bytes of a file specification.
ercRet	The status code to be returned to the calling process.
pbFilespec	The memory address of a string of bytes containing a file specification.
pDataRet	The memory address of an area into which data is to be returned to the calling process.
ppDataRet	The memory address of a 4-byte memory area into which the memory address of a data item is to be returned to the calling process.
pRq	The memory address of a request block.
psDataret	The memory address of 2-byte memory area into which the size of a data item is to be returned.
sData	The size (in bytes) of a data area.
sDataMax	The maximum size (in bytes) of a data area.
ssDataRet	The size of the area into which the size of a data item is to be returned.

Memory Access

The memory access routine provides a way for applications to make direct calls to the graphics memory board regardless of the type of BUS the hardware uses (i.e., FBUS, XBUS, MultiBus).

The one memory access routine is MapGraphicsWindow.

MapGraphicsWindow

MapGraphicsWindow.Obj may be linked with an application in order to provide the application with CPU addressing to the graphics memory board. MapGraphicsWindow.Obj is located in the <sys> directory on the volume on which you initially installed the software.

Procedural Interface

MapGraphicsWindow(*ppGraphicsBoardRet*): *ErcType*

pGraphicsBoard	points to a doubleword that contains the start address of the window of the graphics memory.
ppGraphicsBoardRet	is the memory address of a double word where the start address of the window of the graphics memory is returned.

Graphics Manager Requests

The Graphics Manager service requests are grouped by function into the following five categories.

1 Control Procedures

ClearScreen
InitAdditionalGraphicsScreen
InitScreenGraphics
ReturnGraphicsScreen
SetCommandScreen
SetVisibleScreen
TurnOffGraph ics
TurnOffGraphicsColor
TurnOnGraphics
TurnOnGraphicsColor

2 Vector and Arc Manipulation Procedures

ClearScreenRectangle
ClearPixelScreenRectangle
DrawScreenArc
DrawScreenLine
DrawPixelScreenLine
FillScreenRectangle
FillPixelScreenRectangle
GetDrawAttrInfo
GetRasterInfo
GetVDIViewport
LoadSoftPattern
ReadPixelColor
SetDrawDestinationPlane
SetMonoOrColorDrawMode
SetScreenColor
SetScreenDrawingMode
SetScreenLineType

3 Color Procedures

LoadColor
LoadColorMapper
SetColorMapper

4 Raster Procedures

DoRasterOp
DoRasterOpText
QueryLastRasterText
SetRasterClipping
SetRasterDestination
SetRasterDestinationPlane
SetRasterFont
SetRasterPattern
SetRasterSource
SetRasterSourcePlane
SetRasterTextMode

5 Buffering Procedures

DrawScreenBuffer

Control Procedures

Control procedures handle the screen display control functions. Two such functions are setting the visible screen and the command screen. These procedures initialize graphics pages and return them when completed. A graphics page is the memory that contains the bitmap for the graphics screen (See Glossary). Control procedures are also used to display the visible screen and to clear the video display screen.

Multiple Graphics Pages

Both the B 27 and the B 22 have two graphics pages. In the B 27, each graphics page can be independently allocated. Two different applications running concurrently in a multi-partition environment can each control a graphics page. A single application can also control both graphics pages. Only one graphics page, in either case, can be mapped to the screen. The graphics page that controls the screen is the foreground page and the other graphics page is the background page. To allocate a second graphics page, use the function InitAdditionalGraphicsScreen and return the graphics page by using the function ReturnGraphicsScreen.

The B 22 allocates both graphics pages together. Only one application, therefore, can use both graphics pages at a time. InitScreenGraphics initializes both graphics pages.

Graphics pages should be returned to the system when an application is finished with them. Use ReturnGraphicsScreen to return each page independently within an application.

When a task terminates, any graphics pages allocated to the task are reclaimed by the system.

On the B 27, pages are allocated in sequential order (0, 1) but may be returned in any order. The current command screen is the page returned. When a page is returned, any pages with higher index will have its index decremented, and if the visible screen points to it, the pointer will be changed to reflect the new ordering. If the current visible screen is the current command screen, the visible screen will be set to screen zero, and the display will be blanked until you perform a TurnOnGraphics. If both pages are returned, then the application ceases to be a graphics task and another InitScreenGraphics must be called to allow work to continue. Page zero will become the new command screen.

There are ten control procedures:

- ClearScreen
- InitAdditionalGraphicsScreen
- InitScreenGraphics
- ReturnGraphicsScreen
- SetCommandScreen
- SetVisibleScreen
- TurnOffGraphics
- TurnOffGraphicsColor
- TurnOnGraphics
- TurnOnGraphicsColor

ClearScreen

ClearScreen clears the current command screen to the background color.

Procedural Interface

ClearScreen: *ErcType*

ErcType:

0: No Error
7601: Graphics not initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C075h

InitAdditionalGraphicsScreen

InitAdditionalGraphicsScreen first tests if a graphics screen is available. If the screen is available, it is allocated to the requesting process, clears the display memory to the background color, and sets the command screen to this memory page. If both graphics pages are currently allocated, it will return an error.

Pages are allocated in sequential order, i.e., InitScreenGraphics allocates page zero, and the first call to InitAdditionalGraphicsScreen allocates page one.

Procedural Interface

InitAdditionalGraphicsScreen: *ErcType*

ErcType:

0:	No Error
7607:	No Graphics Screens Left
7601:	Graphics Not Initialized
7802:	Function Not Available

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C071h

InitScreenGraphics

InitScreenGraphics clears the entire display memory to the background color and resets the default values. If a page of graphics has not already been allocated to this task, InitScreenGraphics allocates a page of graphics or returns an error indicating there are no pages left.

Procedural Interface

InitScreenGraphics: *ErcType*

ErcType:

0: No Error
7607: No Graphics Screens Left

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C070h

ReturnGraphicsScreen

ReturnGraphicsScreen returns the current command screen to the Graphics Manager. If this screen was the only screen allocated to the requesting task, graphics is deinitialized for the task (i.e., an InitScreenGraphics must be issued before any other Graphics Manager call is made) and the graphics display is turned off. Any page with index greater than the index of the command screen will have its index decremented. If the visible screen points to one of these screens, its index will also be decremented.

Procedural Interface

ReturnGraphicsScreen: *ErcType*

ErcType

0:	No Error
7601:	Graphics Not Initialized
7802:	Function Not Available on this workstation.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C072h

SetCommandScreen

SetCommandScreen is used only in the graphics workstation applications which allocate more than one graphics page. It specifies which screen is to be the current command screen. Subsequent operations, such as drawing commands, that affect the display memory will use the screen specified.

Acceptable values for B 22 and B 27 workstations are 0 and 1.

Procedural Interface

SetCommandScreen (*iCommandScreen*): *ErcType*

iCommandScreen: Specifies which of the graphics pages is to be used for the command screen.

ErcType:

0:	No Error
7601:	Graphics Not Initialized
7802:	Function Not Available on this workstation.
7800:	Invalid Command Screen Number

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntlnfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C076h
12	iCommandScreen	2	

SetVisibleScreen

SetVisibleScreen is used only in graphics workstation applications that allocate more than one graphics page. It specifies which screen is to be the current visible screen. The visible screen is the one that can be displayed by TurnOnGraphics. The visible screen and the command screen are independent of each other. One screen may be both the visible and command screen simultaneously, depending on the subsequent functions to be performed.

Acceptable values for B 22 and B 27 workstations are 0 and 1.

Procedural Interface

SetVisibleScreen (*iVisibleScreen*): *ErcType*

iVisibleScreen: Specifies which of the graphics pages is to be used for the visible screen.

ErcType:

0:	No Error
7601:	Graphics Not Initialized
7802:	Function Not Available on this workstation.
7801:	Invalid Visible Screen Number

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C077h
12	iVisibleScreen	2	

TurnOffGraphics

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

Procedural Interface

TurnOffGraphics:*ErcType*

ErcType:

0: No Error
7601: Graphics Not Initalized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C078h

TurnOffGraphicsColor

No action on monochrome system. Causes color system to draw monochromatically. Reverses the effect of TurnOnGraphicsColor.

Procedural Interface

TurnOffGraphicsColor:*ErcType*

ErcType:

0	No Error
7601	Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C079h

TurnOnGraphics

TurnOnGraphics displays the screen that has been set as the visible screen. Reverses the effect of TurnOffGraphics.

Procedural Interface

TurnOnGraphics:*ErcType*

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C07Ah

TurnOnGraphicsColor

Has no effect on monochrome workstations. Reverses the effect of TurnOffGraphicsColor.

Procedural Interface

TurnOnGraphicsColor:*ErcType*

ErcType:

0:	No Error
7601:	Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntlnfo	1	0
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C07Bh

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 Graphics Manager 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.

Vectors and arcs are plotted by calculating a line between the specified endpoints. The legal limits for drawing are given in Appendix C.

There are seventeen vector and arc manipulation procedures:

- ClearScreenRectangle
- ClearPixelScreenRectangle
- DrawScreenArc
- DrawScreenLine
- DrawPixelScreenLine
- FillScreenRectangle
- FillPixelScreenRectangle
- GetDrawAttrInfo
- GetRasterInfo
- GetVDIViewpoint
- LoadSoftPattern
- ReadPixelColor
- SetDrawDestinationPlane
- SetMonoOrColorDrawMode
- SetScreenColor
- SetScreenDrawingMode
- SetScreenLineType

ClearScreenRectangle

ClearScreenRectangle clears a rectangular area on the current command screen. The coordinates for the lower left corner of the rectangle are entered as 16-bit words using virtual coordinates. The height and width of the rectangle are also entered as words in relation to the virtual coordinate system.

Procedural Interface

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

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

ErcType:

0: No Error
7601: Graphics Not Initialized
7810: Invalid Clear Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	8
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C07Fh
12	wXStart	2	
14	wYStart	2	
16	wWidth	2	
18	wHeight	2	

ClearPixelScreenRectangle

ClearPixelScreenRectangle clears a rectangular area on the current command screen. The coordinates for the lower left corner of the rectangle are entered as 16-bit words using real (physical) coordinates. The height and width of the rectangle are also entered as words in relation to the coordinate system.

Procedural Interface

ClearPixelScreenRectangle (*wXStart, wYStart, wWidth, wHeight*): *ErcType*

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

0: No Error
7601: Graphics Not Initialized
7810: Invalid Clear Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	8
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C09Eh
12	wXStart	2	
14	wYStart	2	
16	wWidth	2	
18	wHeight	2	

DrawScreenArc

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

Figure 7-1 illustrates the values that are used for the parameters in DrawScreenArc.

The coordinate position for the start of the arc is given in current virtual coordinates.

The maximum size arc that can be drawn by this operation is an octant of a circle. To draw a longer arc, DrawScreenArc must be called again until the intended size is reached. Figure 7-2 illustrates the drawing directions.

On a B 26, and B 27/B 28, DrawScreenArc draws arcs using actual pixel coordinates rather than the adjusted screen coordinates. For this reason, arcs are stretched vertically because there are more pixels per inch horizontally than vertically.

Procedural Interface

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

ErcType

wX,wY: Specifies the coordinate position for the start of the arc.

wDir: Specifies the drawing direction

wD: Specifies the (radius-1), in bits

wD2: Specifies 2*(radius-1), in bits

wDC: Specifies radius * sin (phi), rounded up, where phi = the angle between the axis and the far end of the arc.
wDC<= (radius/sqrt(2)) 0 < phi < (pi/4)

wDM: Specifies radius * sin (theta), rounded down, where theta = the angle between the axis and the start of the arc. 0<=wDM<=wDC 0 < theta < (pi/4)

ErcType:

0: No Error
7601: Graphics Not Initialized
7811: Arc Parameters Invalid

Figure 7-1 **Determining Arc Length**

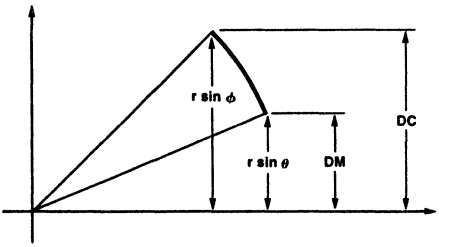
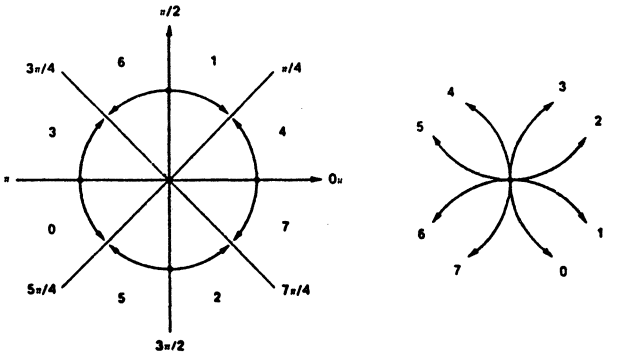


Figure 7-2 **Drawing Directions (Angles in Radians)**



Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	14
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C080h
12	wX	2	
14	wY	2	
16	wDir	2	
18	wD	2	
20	wD2	2	
22	wDC	2	
24	wDM	2	

DrawScreenLine

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.

Parameters are in virtual coordinates.

Procedural Interface

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

wX1,wY1: Specify the coordinate position of the beginning of the vector

wX2,wY2: Specify the coordinate position of the end of the vector

ErcType:

0: No Error
7601: Graphics Not Initialized
7812: Invalid Line Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	8
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C081h
12	wX1	2	
14	wY1	2	
16	wX2	2	
18	wY2	2	

DrawPixelScreenLine

DrawPixelScreenLine 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.

Parameters are in real (physical) coordinates.

Procedural Interface

DrawPixelScreenLine(*wX1, wY1, wX2, wY2*): *ErcType*

wX1,wY1: Specify the coordinate position of the beginning of the vector

wX2,wY2: Specify the coordinate position of the end of the vector

ErcType:

0:	No Error
7601:	Graphics Not Initialized
7812:	Invalid Line Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	8
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C09Fh
12	wX1	2	
14	wY1	2	
16	wX2	2	
18	wY2	2	

FillScreenRectangle

FillScreenRectangle fills a rectangle in the current command screen with a pattern. The values for the coordinates that define the rectangle are entered in virtual coordinates. The pattern is also specified as a parameter. (See Appendix B for an illustration of the fill patterns.)

For B 22 workstations, you cannot clear a rectangle by calling fill type '1' (clear pattern). You must use ClearScreenRectangle instead.

Procedural Interface

FillScreenRectangle(*wX1, wY1, wX2, wY2, wFilltype*): *ErcType*

wX1,wY1: Specify the lower left corner of the rectangle

wX2,wY2: Specify the upper right corner of the rectangle

wFillType: Specifies the pattern that is to be used to fill the rectangle.
0–5 = the fill patterns

Appendix B illustrates the various fill patterns available.

ErcType:

0: No Error
7601: Graphics Not Initialized
7813: Invalid Fill Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	10
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C082h
12	wX1	2	
14	wY1	2	
16	wX2	2	
18	wY2	2	
20	wFillType	2	

FillPixelScreenRectangle

FillPixelScreenRectangle fills a rectangle in the current command screen with a pattern. The values for the coordinates that define the rectangle are entered in virtual coordinates. The pattern is also specified as a parameter. (See Appendix B for an illustration of the fill patterns.)

For B 22 workstations, you cannot clear a rectangle by calling fill type '1' (clear pattern). You must use ClearPixelScreenRectangle instead.

Procedural Interface

FillPixelScreenRectangle(*wX1, wY1, wX2, wY2, wFilltype*): *ErcType*

wX1,wY1: Specify the lower left corner of the rectangle
wX2,wY2: Specify the upper right corner of the rectangle
wFillType: Specifies the pattern that is to be used to fill the rectangle.
0?n5 = the fill patterns

Appendix B illustrates the various fill patterns available.

ErcType:

0: No Error
7601: Graphics Not Initialized
7813: Invalid Fill Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	10
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C0A0h
12	wX1	2	
14	wY1	2	
16	wX2	2	
18	wY2	2	
20	wFillType	2	

GetDrawAttrInfo

GetDrawAttrInfo returns the current color, line type and drawing mode from the Graphics Control Block.

Procedural Interface

GetDrawAttrInfo(*pGCBAttr*): *ErcType*

pGCBAttr: Points to the memory address where the drawing information is to be returned in the following format:

bColor: Specifies the current color(1 byte)

bLineType: Specifies the current line type(1 byte)

bDrawingMode: Specifies the current drawing mode(1 byte)

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C088h
12	reserved	6	
18	pbGCBAttr	4	
22	cbGCBAttr	2	3

GetRasterInfo

GetRasterInfo returns information about the raster of the workstation. The length of the returned raster information depends on the number (and types) of bitmap planes per graphics page the system uses. The length of the return raster information is 24 bytes, which will provide up to three memory planes. If you need additional hardware information, use the BTOS call QueryVidHdwr.

Note: *Because all machines have different virtual pixel resolutions, make your programs as device-independent as possible; use GetRasterInfo to determine virtual and physical pixel resolutions.*

See Appendix C, "Graphics Manager Virtual and Physical Pixel Resolution," for information on each workstation's physical resolutions.

Procedural Interface

GetRasterInfo (*pbRasterInfo*): *ErcType*

pbRasterInfo:	Points to the memory address where the raster information is to be returned in the following format:
cntPlanes	(2 bytes) number of memory planes for this graphics system
wBytesPerLine	(2 bytes) the number of bytes of data per physical raster line
wWidth	(2 bytes) the number of pixels across the bitmap horizontally (in current 80- or 132- mode)
wHeight	(2 bytes) the number of pixels across the bitmap vertically (in current 80- or 132- mode)
wVirtualWidth	(2 Bytes) screen width in virtual pixels (current 80- or 132- mode)
wVirtualHeight	(2 Bytes) screen height in virtual pixels (current 80- or 132- mode)
pPlanei	(4 bytes) pointer to plane i of the bitmap (there are cntPlanes pointers returned by this call)

ErcType:

0: No Error

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C08Ch
12	reserved	6	
18	pbRasterInfo	4	
22	cbRasterInfo	2	24

GetVDIViewport

GetVDIViewport returns the x and y size in terms of the real coordinate system where 1 equates to 32767.

Procedural Interface

GetVDIViewport (*pXSize,pYSize*)*: ErcType*

pXSize Points to the memory address where the X size is to be returned. The X size is always 32767 (1).

pYSize Points to the memory address where the Y size is to be returned. The Y size is a fraction of the X size determined by the real coordinate system.

ErcType:
0: No Error
7601: Graphics not initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C0A1h
12	reserved	6	
18	pXSize	4	
24	pYSize	4	

LoadSoftPattern

LoadSoftPattern provides for the use of user-defined line types. This procedure defines one single line type. Besides a parameter to identify the line type, there is a word to hold the 16-bit value itself. To use a line type that has been defined with this procedure, the value for the line type parameter in SetScreenLineType 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.

On B 22 workstations, the only acceptable values are 0 to 15.

Procedural Interface

LoadSoftPattern (*iPattern, wPattern*): *Erctype*

iPattern: Specifies the line type. The acceptable values are 0 to 7

wPattern: Specifies the 16-bit pattern

ErcType:

0: No Error
7601: Graphics Not Initialized issued first
7814: Invalid Pattern Index

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	4
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C083h
12	iPattern	2	
14	wPattern	2	

Specifying wPattern in LoadSoftPattern

Example

Erc = LoadSoftPattern *(0, 0EAEAh);*

0 is the pattern number

0EAEAh is the pattern

The pattern (0EAEAh) is composed as follows:

Load position	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Set(1)/Reset(0)	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	0
	E				A				E				A			

This pattern is shown in Figure 7-3.

Figure 7-3 **Sample Pattern**

ReadPixelColor

Returns the color of a pixel or the current command screen. On a monochrome system, the result will be either 0 (off) or 1 (on). B 26, and B 27/B 28 use virtual coordinates for the (x,y) point.

Procedural Interface

ReadPixelColor (*wX, wY, pbColor*):*ErcType*

wX,wY: Specify the pixel in virtual coordinates.
pbColor The address of a byte to return the color of the specified pixel. If the pixel is out of range, the result is undefined.

ErcType:

0: No Error
7601: Graphics Not Initialized
7802: Function Not Available on B 21
7815: Invalid Pixel Address

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C084h
12	wX	2	
14	wY	2	
16	reserved	2	
18	pbColor	4	
22	cbColor	2	1

SetDrawDestinationPlane

SetDrawDestinationPlane selects the drawing planes. If the parameter passed is an 8, then fMonoMode drawing occurs as in normal operations. If the parameter is 1, 2, or 4 then the plane selected is 0,1, or 2 respectively. Subsequent drawing only occurs on the selected plane.

This call does not affect monoplane machines (for example, B 22, B 27 mono).

Procedural Interface

SetDrawDestinationPlane(*wDestMask*): *ErcType*

wDestMask: Specifies the plane to be selected for subsequent drawing commands.

ErcType:

0:	No Error
7601:	Graphics Not Initialized
7646:	Bad Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C0A2h
12	wDestMask	2	

SetMonoOrColorDrawMode

SetMonoOrColorDrawMode selects drawing in 'mono' mode or color mode by calling SetDrawDestinationPlane with either 1 (for mono) and 8 (for color). Drawing in 'mono' mode means drawing occurs only on plane 0, and drawing in color mode means drawing occurs on all three planes.

This call does not affect monoplane machines (for example, B 22, B 27 mono).

Procedural Interface

SetMonoOrColorDrawMode(fMono): *ErcType*

fMono: Specifies whether to draw on one or three planes. If fMono is False, then one plane; otherwise, all three planes.

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C0A3h
12	fMono	2	

SetScreenColor

SetScreenColor sets the foreground drawing color for vector, rectangle, arc, and fill commands. The default screen drawing color is color 1. For a monochrome system, the only acceptable values are 0 and 1.

Procedural Interface

SetScreenColor(*iScreenColor*): *ErcType*

iScreenColor: Specifies the color to be used in subsequent commands, in the range 0 to n, where n is the number of colors (gray levels) this hardware supports. For monochrome systems, there are only two gray levels available, 0 or 1.

ErcType:

0:	No Error
7601:	Graphics Not Initialized
7816:	Invalid Screen Color

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C085h
12	iScreenColor	2	

SetScreenDrawingMode

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 Section 4 for detailed information about drawing modes.

Procedural Interface

SetScreenDrawingMode(*iDrawingMode*): *ErcType*

iDrawingMode: Specifies the drawing mode to be set.

0 = set
1 = clear
2 = complement
3 = replace

ErcType:

0:	No Error
7601:	Graphics Not Initialized
7817:	Invalid Drawing Mode

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C016h
12	iDrawingMode	2	

SetScreenLineType

SetScreenLineType 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 subsection "Drawing Attributes" for detailed information about line types.

Procedural Interface

SetScreenLineType(*iLineType*): *ErcType*

iLineType: Specifies the line type
0 to 7 = the standard line types (see Figure 4-1)
8 to 15 = user defined line types (see LoadSoftPattern)

ErcType:

0: No Error
7601: Graphics Not Initialized
7818: Invalid Line Type

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C087h
12	iLineType	2	

Color Procedures

Color procedures are available only on the B 21 and B 25 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 primary colors, and any eight of these 64 possibilities can be displayed on the screen at one time.

An 8-byte memory workarea is used to specify the eight colors and load the color mapper (on the B 21 graphics control board or the B 25 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 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 7-1 shows the position of the 2-bit color setting 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, hexadecimal notation and color. The colors in the default palette are composed of combinations of various intensities red, green, and blue.

Table 7-1 **The Default Color Palette**

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

There are three color procedures:

- LoadColor
- LoadColorMapper
- SetColorMapper

LoadColor

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*

iMapper: Specifies the color mapper that is to be loaded:
0 = first mapper
1 = second mapper

iColor: Specifies which color in the palette is to be modified. These values range from 1 through 8.

bColor: Specifies the new value of the color.

ErcType:

0:	No Error.
7601:	Graphics Not Initialized.
7830:	Invalid Color Mapper.
7831:	Invalid Mapper Index.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntlnfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C098h
12	iMapper	2	
14	iColor	2	
16	bColor	2	

LoadColorMapper

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 color mappers on the graphics control board 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, pbColors*): *ErcType*

iMapper: Specifies the color mapper that is to be loaded:
0 = first mapper
1 = second mapper

pbColors: Specifies the new palette to be loaded into the specified color mapper. pbColors should contain as many bytes as there are colors in the palette.

ErcType:

0: No Error
7601: Graphics Not Initialized
7830: Invalid Color Mapper

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C099h
12	iMapper	2	
14	reserved	4	
18	pbColors	4	
22	cbColors	2	8

SetColorMapper

SetColorMapper specifies which of the color mappers on the graphics control board 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.

This routine has no effect on workstations which are either monochrome or have only one color mapper

Procedural Interface

SetColorMapper(*iMapper*): *ErcType*

iMapper: Specifies which color mapper is to be current:
0 = first mapper
1 = second mapper

ErcType:

0: No Error
7601: Graphics Not Initialized
7830: Invalid Color Mapper

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	2
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C09Ah
12	iMapper	2	

Color Text

If you want to use color with the text attributes, refer to PutFrameAttributes in *BTOS Reference Manual Volumes 1 and 2.*

Raster Procedures

Bit manipulation procedures provide an alternative to vector drawing. Rectangular areas of any size can be filled with light, dark, or halftone patterns of bits. They can also be copied from bitmap to bitmap or moved about within the same bitmap. When display memory is used as the destination bitmap, rectangles can be written, copied, and moved around the screen very quickly to support such functions as scrolling and animation.

Destination Rectangles

Bit manipulation procedures produce images on the screen by modifying a rectangular area of display memory known as the *destination*. A destination's contents is a function of the following elements:

- Its previous contents
- The contents of another rectangle of equal size (the *source*)
- A 16 word area (the *pattern*)

The destination's bits can be combined with the bits in the source (and/or the pattern) to produce a variety of effects. The exact effect is determined by the *source operation* and the *destination operation* (user-specified operations).

The source operation involves the source rectangle and the pattern. The destination operation involves the result of the source operation and the bits in the destination rectangle. The result of both operations is a new set of the bits in the destination rectangle.

As an option, the destination rectangle may be clipped to the interior of a user-defined clipping box. When writing to display memory, this feature could be used to restrict the effects of raster operations to a particular portion of the screen.

Note: *Raster procedures need not only be used to write to the screen. Any area of memory may be written to (or read from) using these procedures.*

Source and Destination Bitmaps

The area from which the source rectangle is taken is the *source bitmap*. The area to which the destination rectangle is written is the *destination bitmap*.

Raster procedures may be used to copy any rectangular image as follows:

- From memory to screen
- From memory to memory
- From screen to memory
- From one portion of the screen to another

Note the following examples:

- The source and destination bitmaps could be the same, and the source and destination rectangles could even overlap.
- The source bitmap could be in main?cpu (non?display) memory, and the destination bitmap could be in display memory. For example, you could copy an icon from the source bitmap (main memory) to the destination bitmap (screen).
- The source bitmap could be in display memory, and the destination bitmap could be in main?CPU memory. For example, you could save a portion of the screen image for later use.

Note: *With respect to color systems, rectangles may be moved from one plane of display memory to another (with restrictions which will be noted later). The B 21 workstation is an exception in that display memory is not accessible. Thus, raster operations cannot be used to affect B 21 screens.*

Text Manipulation

Raster text-manipulation procedures enable you to display multiple user-defined, bit-mapped fonts. Characters within any given font may be of any height and width. The user-supplied font specification can be used to convert each string of numeric character codes to its bitmap representation.

Characters are painted in the current drawing mode: Each character's bitmap representation is painted in its corresponding portion of the destination bitmap. The format in which these fonts must be created is described later.

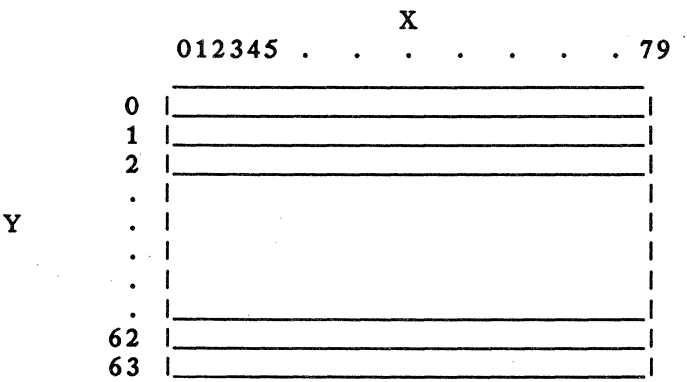
Bit-Addressable Pixel Coordinates

All raster operations use the same bit-addressable pixel coordinate system that views the bitmap as a logical rectangle.

Note: *Source and destination rectangles are specified in terms of these coordinates.*

Each rectangle is described in terms of the (x,y) location of its upper left corner along with its width and height in bits. The coordinates (0,0) map to the upper left corner of the bitmap, while (max X, max Y) map to the lower right corner.

Note: *Neither max X nor max Y may exceed 32767, because the raster routines will exhibit unpredictable behavior.*



Bitmaps are defined by four parameters:

- Memory address
- Width
- GWS (graphics) flag
- Plane

Memory address, which must have an offset of zero, defines the location of an area of memory that is reserved for the bitmap.

Width, which must be divisible by 2, gives the number of bytes in one scan line of the map. Note that width of the map is measured in pixels divided by 8.

The *GWS flag* has meaning only for a B 22 workstation; it indicates whether or not the bitmap lies in graphics (GWS) memory. This is important because on the B 22, the bitmap addresses (used to address display memory through raster operations) are not main-cpu addresses. Rather, they are the addresses that the graphics-board cpu would use to address the bitmap (see "Raster Usage Notes" later on in this section).

A numeric value indicates the graphics *plane* of display memory in which the bitmap lies:

0 Bitmap in non-display memory

1 Bitmap on first graphics plane

2 Bitmap on second graphics plane

4 Bitmap on third graphics plane

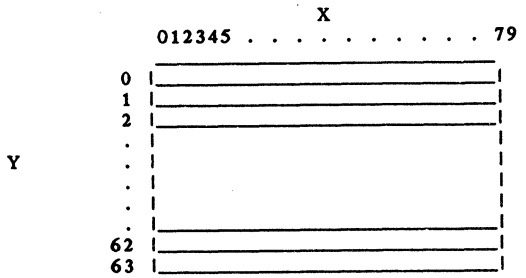
8 A plane value of 8 is a special indicator for raster text routines. It indicates that the characters should be painted on all planes of display memory. On the B 27 mono, B 21, and B 22, this plane value is ignored since planes are meaningless on these machines.

As an example, examine a bitmap with the following characteristics:

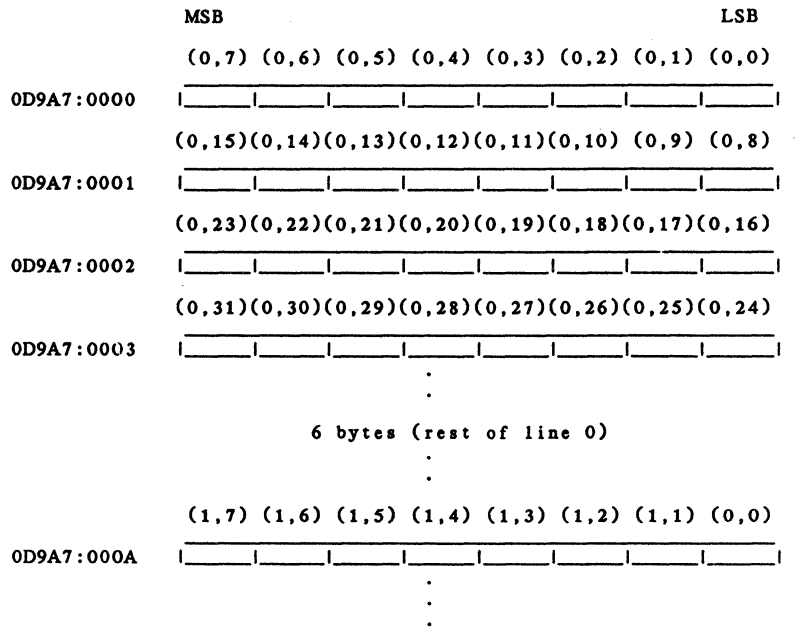
- Plane = 0
- On a B 22, the GWS flag = 0
- Dimensions: 80 bits wide by 64 bits high

A bitmap having the above characteristics does not lie in display memory. Its width is 10 (80 bits divided by 8). To determine how much memory is required to represent this map, multiply width by height: 640 bytes. To obtain a memory address having an offset of zero, call AllocMemorySL. The address that it returns is *0D9A7:0000h*.

The logical view of this bitmap is presented below. Source, destination, and clipping rectangles are specified in terms of these coordinates.



While this is the view of the world that the raster routines present, the bitmap is actually organized in memory as a sequence of bytes using the following bit-to-coordinate correspondence:



The physical (as opposed to logical) layout of the bitmap must be remembered if you wish to construct an image (for example, an icon) in memory for later transfer to display memory.

Caution: *Raster procedures perform minimal error checking. Bad parameters may cause fatal errors.*

There are eleven raster procedures, which are described in the following pages.

- DoRasterOp
- DoRasterOpText
- QueryLastRasterText
- SetRasterClipping
- SetRasterDestination
- SetRasterDestinationPlane
- SetRasterFont
- SetRasterPattern
- SetRasterSource
- SetRasterSourcePlane
- SetRasterTextMode

DoRasterOp

DoRasterOp performs the actual manipulation of the rectangles of bits. The parameters passed are the pixel coordinates of the source rectangle in relation to the source bitmap, the pixel coordinates of the destination rectangle in relation to the destination bitmap, the height and width of the rectangles, and indications of the source and destination operations. Other information, such as the clipping parameters and the addresses of the bitmaps, must have previously been recorded by SetRasterSource, SetRasterDestination, SetRasterPattern, SetRasterClipping, SetRasterDestinationPlane, and SetRasterSourcePlane.

If raster clipping is enabled, the destination rectangle is clipped to the interior of the clipping box defined by a previous call to SetRasterClipping.

On a B 22 workstation, if the pattern lies in graphics memory, the source and destination bitmaps must also. If this is not the case, DoRasterOp will not function correctly. This restriction is a consequence of the fact that if all operands are present in graphics memory, the raster operation can be passed off to the graphics control board. The inverse restriction also applies (i.e. if both source and destination bitmaps lie in graphics memory, the pattern must also).

Procedural Interface

DoRasterOp (*wxSrc, wySrc, wxDst, wyDst, wdx, wdy, wopSrc, wopDst*): *ErcType*

wxSrc, wySrc: Pixel coordinates of the upper left corner of the source rectangle relative to the bitmap set by SetRasterSource.

wxDst, wyDst: Pixel coordinates of the upper left corner of the destination rectangle relative to the bitmap set by SetRasterDestination.

wdx, wdy: Width and height of the destination in bits.

wopSrc: Specifies the function that is to be performed on the source.
0 ==> source is source rectangle
1 ==> source is pattern
2 ==> source is pattern AND source rectangle
3 ==> source is pattern OR source rectangle
4 ==> source is pattern XOR source rectangle

wopDst: Specifies the function that is to be performed on the destination.
0 ==> dest <- source
1 ==> dest <- source AND dest rectangle
2 ==> dest <- source OR dest rectangle
3 ==> dest <- source XOR dest rectangle
4 ==> dest <- NOT source AND dest rectangle

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	16
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C08Bh
12	wxSrc	2	
14	wySrc	2	
16	wxDst	2	
18	wyDst	2	
20	wdx	2	
22	wdy	2	
24	wopSrc	2	
26	wopDst		

DoRasterOpText

DoRasterOpText converts an array of character codes to their bitmap representations and paints them onto the destination bitmap. This procedure uses the font supplied in SetRasterFont. The arguments passed are: a pointer to the list of character codes to be painted, the numbers of characters to paint, the x and y coordinates where painting should begin (relative to the destination bitmap), and an x clipping limit. The bitmap representations of the characters are then painted on the destination bitmap and plane (set by SetRasterDestination), beginning at the point given. The characters are painted in the current raster text mode (set by SetRasterTextMode). If the destination plane is 8, then the character is painted on all planes of display memory.

The x clipping limit passed in the DoRasterOpText call is used as the right text boundary. Any character which would cause painting to occur to the right of this limit is not displayed.

Otherwise, painting begins and continues until:

1 All characters have been painted.

2 The right x clipping limit is encountered.

When painting halts for either of the above reasons, a count of characters successfully written is stored for possible retrieval by QueryLastRasterText.

Note that it is the upper left corner of the first character which is drawn at the indicated (x,y).

On the B 22, if both the font and the destination bitmap lie in graphics memory, the painting operation will be passed to the graphics control board. In this situation, painting will occur in Set mode regardless of the current raster text mode. Further, no count of characters written will be recorded for QueryLastRasterText. Rather, the invalid value (-1) will be recorded as an indicator that this situation has arisen.

Procedural Interface

DoRasterOpText(*pbText, cbText, wXStart, wYStart, wXLim*): *ErcType;*

pbText, cbText:	Specify the memory address and length of the array of characters to be converted.
wXStart, wYStart:	Specify the position on the destination bitmap where painting should begin.
wXLim:	The absolute x-coordinate position where painting must stop, even if there are more characters to draw.

ErcType:

0:	NoError
7601:	Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C094h
12	wXStart	2	
14	wYStart	2	
16	wXLim	2	
18	pbText	4	
20	cbText	2	

QueryLastRasterText

QueryLastRasterText allows a user to inquire how many characters were painted by the last DoRasterOpText call. If no previous DoRasterOpText call has been made, the value returned is unpredictable.

One argument is required, a pointer to a word where the character count will be written.

Procedural Interface

QueryLastRasterText(*pWord***)**: *ErcType*

pWord: Pointer to a word where the character count will be written.

ErcType:

0: No Error
7601 Graphics Not Initialized
zzzzz

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C093h
12	reserved	6	
18	pWord	4	
22	constant	2	2

SetRasterClipping

SetRasterClipping controls the clipping action taken by DoRasterOp. Its arguments are a flag indicating whether clipping is active and a description of a clipping box. If clipping is in effect DoRasterOp clips all operations to the interior of the clipping box.

Note: *The edges of the clipping box are considered "interior".*

Procedural Interface

SetRasterClipping(*fOn, wX, wY, wDx, wDy*): *ErcType*

fOn: Flag indicated whether raster clipping should be performed.

wX,wY: The x and y coordinates of the upper left corner of the clip box.

wDx,wDy: The width and height of the clip box.

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	10
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C092h
12	fOn	2	
14	wX	2	
16	wY	2	
18	wDx	2	
20	wDy	2	

SetRasterDestination

SetRasterDestination defines the location and dimension of the destination bitmap. This procedure also defines, on a B?22 only, whether the destination bitmap lies in GWS (graphics) memory. Subsequent raster mode operations will use the specified location for the destination.

This procedure requires three parameters: the bitmap address, the width in bytes of the bitmap, and a flag indicating whether the address is a GWS address. This flag has meaning only on a B-22 and is ignored on all other machines.

The bitmap address should have a 0 as the offset portion, or the raster routines may function improperly. Unexpected results will occur on a B-22 if the GWS flag is non-zero and the bitmap address passed is in fact not GWS relative. Similarly, the destination bitmap must be no larger than 64K, or the raster routines will not function correctly.

Procedural Interface

SetRasterDestination(*pBmDest, cbLineBm, fGWS*): *ErcType*

pBmDest: Address of the destination bitmap.

cbLineBm: Width of the bitmap in bytes.

fGWS: B 22 only - flag set if destination in graphics memory.

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C08Dh
12	fGWS	2	
14	reserved	4	
18	pBmDest	4	
22	cbLineBm	2	

SetRasterDestinationPlane

SetRasterDestinationPlane defines the plane of graphics memory upon which the destination bitmap lies. Valid planes values are: 0 - non-graphics memory, 1 - 1st plane, 2 - 2nd plane, 4 - 3rd plane, 8 - all planes.

A destination plane of 8 causes DoRasterOpText to paint characters on all planes of graphics memory simultaneously. DoRasterOp treats a desination plane of 8 as though it were a 0.

Note that on the B 27 mono, B 21, and B 22 this call is ignored, since planes have no meaning on these machines.

Procedural Interface

SetRasterDestinationPlane(*wPlane*): *ErcType*

wPlane: Plane of graphics memory on which destination bitmap lies.

ErcType:

0: No Error
7601: Graphics Not Initialized
7646: Bad Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C08Eh
12	wPlane	2	

SetRasterFont

SetRasterFont specifies the location of the font to be used in subsequent calls to DoRasterOpText. Two parameters are passed, the address of the font, and a flag indicating whether the font lies in graphics memory. This flag is ignored on all machines except the B 22.

The address of the font should have 0 as its offset portion, or the raster text routines may function improperly. No font may be greater than 64K in size.

Procedural Interface

SetRasterFont(*pFont, fGWS*): *ErcType*

pFont: Pointer to the font.

fGWS: B 22 only - is font address GWS (graphics board) relative.

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C095h
12	reserved	6	
18	pFont	4	
22	wPlane	2	0

SetRasterPattern

SetRasterPattern specifies the location of the 16-word halftone pattern used by DoRasterOp, along with a flag indicating whether the pattern resides in graphics memory. The address of the pattern must be paragraph aligned (i.e. 0 MOD 16). The fGWS flag passed in the call is ignored on all machines except the B 22. If the fGWS flag is set on a B 22, it is assumed that the source and destination bitmaps also lie in GWS (graphics) memory, and all future RasterOps will be passed off to the graphics control board to perform. If this is in fact not the case, unexpected results will ensue.

The pattern is used to describe a pseudo-bitmap of infinite dimension, where the row of the bitmap with y-coordinate Y is the (Y MOD 16)th word of the pattern replicated infinitely. The rectangle of the pseudo-bitmap which is ANDed, ORed, etc., with the source rectangle is defined by the description of the destination rectangle (upper left corner, dx, dy) as applied to the pseudo-map.

For example, given the pattern below (actual representation in memory), the corresponding logical bitmap is as shown.

Pattern at 0DA7:0000h:

```
                          MSB                                LSB
Word at  0DA7:0000  0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0
Word at  0DA7:0002  1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
Word at  0DA7:0004  0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
     .
     .
     .
Word at  0DA7:001E  1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
```

Corresponding logical bitmap:

```
 x    |
----- |0123456 .  .  .
 y   \|
-----------------------------------------------------
 0    |01111100000110000111110000011000011111 .  .  .
 1    |00000001111111110000000111111111000000 .  .  .
 2    |00001111000000000000011110000000000011 .  .  .
      |                    .
      |                    .
      |                    .
15    |01010101010101010101010101010101010101 .  .  .
      |                    .
      |                    .
      |                    .
48    |01111100000110000111110000011000011111 .  .  .
49    |00000001111111110000000111111111000000 .  .  .
50    |00001111000000000000011110000000000011 .  .  .
      |                    .
      |                    .
      |                    .
63    |01010101010101010101010101010101010101 .  .  .
      |                    .
      |                    .
      |                    .
```

Procedural Interface

SetRasterPattern(*pRgwPat, fGWS*): *ErcType*

pRgwPat: Memory address of the pattern.

fGWS: B 22 - flag set if pattern address is display memory address.

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntlnfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C08Fh
12	reserved	6	
18	pRgwPat	4	
22	fGWS	2	

SetRasterSource

SetRasterSource defines the location and dimension of the source bitmap. This procedure also defines, on a B 22 only, whether the source bitmap address is a graphics board address. Subsequent raster mode operations will use the specified location for the source.

The procedure requires three parameters: the bitmap address, the width in bytes of the bitmap, and a flag indicating whether the source lies in GWS (graphics) memory. This flag is ignored on all machines except the B 22.

The bitmap address should have 0 as its offset portion, or the raster routines may function incorrectly. Further, the width specified for the bitmap must be even.

Unexpected results will occur on the B 22 workstation if the fGWS flag is set and the source bitmap address is in fact not GWS relative.Similarly, the source bitmap must be no larger than 64k, or the raster routines will not function correctly.

Procedural Interface

SetRasterSource(*pBmSrc, cbLineBm, fGWS*): *ErcType*

pBmSrc: Address of source bitmap.

cbLineBm: Width of the bitmap in bytes.

fGWS: B 22 only - is the source bitmap address GWS relative.

ErcType:

0: No Error
7601: Graphics Not Initialized

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C090h
12	fGWS	2	
14	reserved	4	
18	pBmSrc	4	
22	cbLineBm	2	

SetRasterSourcePlane

SetRasterSourcePlane defines the plane of graphics memory on which the source bitmap lies. Valid plane values are: 0 - non-graphics memory, 1 - 1st plane of graphics memory, 2 - 2nd plane of graphics memory, and 4 - 3rd plane of graphics memory.

On the B 27 mono, B 21, and B 22, this call is ignored, since planes are meaningless on these systems.

Procedural Interface

SetRasterSourcePlane(*wPlane*): *ErcType*

wPlane: Plane of graphics memory on which source bitmap lies.

ErcType:

0: No Error
7601: Graphics Not Initialized
7646: Bad Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C091h
12	wPlane	2	

SetRasterTextMode

SetRasterTextMode defines the mode in which all future raster text characters will be drawn. Valid modes are: 0 - Set, 1 - Clear, and 2 - Complement.

In Set mode, the bitmap describing each character is ORed with the destination bitmap. In Complement mode it is XORed. In Clear mode the inverse of the character bitmap is ANDed with the destination.

Procedural Interface

SetRasterTextMode(*wMode*): *ErcType*

wMode: Mode in which future raster text will be painted.

ErcType:

0: No Error
7601: Graphics Not Initialized
7646: Bad Parameters

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	0
3	nRespPbCb	1	0
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C096h
12	wMode	2	

Raster Font Format

This section describes the raster font format and the manner in which DoRasterOpText uses it in detail.

The raster font format required by DoRasterOpText contains the following information for each character: the width of the character in bits, the offset from the start of the font to the character's description, and the description itself.

The description of the character begins with information which tells how many bits to skip down and to the right before beginning to paint the character. This allows for the bitmap which defines the character to be left and top justified—if the user wishes—to save space.

Following this information is the bitmap describing the character. This format of this bitmap differs from the "normal" bitmap format. Character bitmaps are organized as a series of "widths," each describing a 16-bit wide slice of the character.

Each width begins with two pieces of header information, the number of unused bits on the right of the width, and the height in bits of the width (cw). This RightZeros field should be zero on all but the last width of the character, where it should be (16 - width of char in bits MOD 16) MOD 16. Note that this may be zero if the character is a mutliple of 16 bits wide. Any bits indicated as unused MUST be zeroed in the bitmap.

Following this header is the bitmap describing the width. This consists of cw words, describing the rows of the width from the top row to the bottom.

An extra width is required at the end of the valid widths as an end-of-character marker. This width consists of a RightZeroBits,cw pair in which cw equals zero.

As an example, let us examine how the following character would be represented.

```
zzzzz
(0,0) @@@@@@@@@@@@@@@@@@@@@@@@
      @@@@@@@@@@@@@@@@@@@@@@@@      . = bit off
      ........................      @ = bit on
      ........................
      @@@@@@@@@@@@@@@@@@@@@@@@
      @@@@@@@@@@@@@@@@@@@@@@@@      (21,5)
```

The above character (an equals sign) is best represented as both top and left justified as shown. We can add 7 bits on the top and 5 on the left by using the skip information stored in the font for the character. Thus, examined on a word-by-word . basis the description of this character would be as follows (all numbers hexadecimal):

```
0507   ;skip 5 down and 7 to right
0600   ;width is 6 bits high, 0 unused on right
FFFF
FFFF
0000   ;bitmap for this width
0000
FFFF
FFFF
0608   ;width is 6 bits high, 8 unused on right
00FF
00FF
0000   ;bitmap for this width
0000
00FF
00FF
0000   ;zero RightZeroBits,cw pair to end char
```

An exact enumeration of the fields of a font, their location, and sizes follows. Fields marked "ignored" are not currently used by the raster text painting routine.

Font Format

Header (occurs once):

Field	Location	Size
size	WORD	The total size of the font in bytes.
dyCell	BYTE	The height of the largest character in the font. Ignored.
dyToBl	BYTE	Distance from the top of the character bitmap cell to the baseline - dyCell minus any descender. Ignored.
mpchrbFe	(256) WORD	An array that maps character codes to the relative bytes of their font entry.
mpchdx	(256) WORD	An array that maps character codes to the width (in bits) of the corresponding character.
grpFe		Font entries as described below.

Font Entry (one per character in the font):

Field	Location	Size
dyFirst	BYTE	The number of raster scan lines to skip before beginning to paint the character (the bitmap is top-justified).
dxFirst	BYTE	The number of bits to skip in the x direction before beginning to paint (the bitmap is left-justified).
widths		Width entries as described below.

Width entries (1 + one per 16-bit width of the character):

Field	Location	Size
rightZeroBits	BYTE	The number of unused bits on the right side of this width (for store optimization).
cw	BYTE	The height (in bits) of this width of the character.
rgwBm	(cw) WORD	The bitmap describing this width of the character.

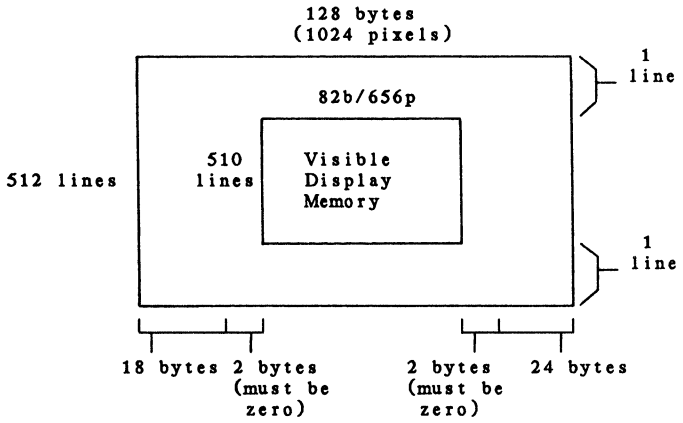
Note that a width entry in which cw equals 0 is used to signal the end of the character.

Raster Usage Notes

This section describes on a per-machine basis how to obtain the parameters needed to use raster routines to access the graphics display.

B 21 Raster operations may not be used to access display memory.

B 22 Use address 8000:0000h to address the first page of graphics memory, 9000:0000h to access the second page. Do not use the address returned by GetRasterInfo. Set fGWS to TRUE. Each page of display memory is a bitmap with the following format:



The upper left corner of visible display memory lies at (160,1). 16 bits on both the right and left sides of the visible portion of display memory MUST BE zero (i.e. don't overwrite this area). The rest of the bitmap, including both visible and nonvisible areas, may be accessed however.

One use for the nonvisible area might be to store the pattern when both the source and destination bitmaps are in graphics memory (as you recall, if both source and dest are in graphics memory on the B 22, the pattern must be also).

B 26: Mono and Color Use GetRasterInfo to obtain the pointer to the appropriate plane. Specify the width of the bitmap as the wBytesPerLine value returned by GetRasterInfo. fGWS is irrelevant. Use SetRasterDestinationPlane and SetRasterSourcePlane to select the source and destination planes you wish.

The first 720 columns and 348 rows are visible display memory. The last 16 rows of the bitmap are not visible, however, and should not be accessed. Unpredictable results will ensue if they are accessed.

B 27: Mono and Color Use GetRasterInfo to obtain the pointer to the appropriate plane. Specify the width of the bitmap as the wBytesPerLine value returned by GetRasterInfo. fGWS is irrelevant. On a color system, use SetRasterSourcePlane and SetRasterDestinationPlane to select the appropriate source and destination planes.

In 80 column mode, the first 720 columns and 480 rows are visible display memory. In 132 column mode, the first 792 columns and 480 rows are visible.

On the B 27, each plane of memory contains two pages of graphics. Direct transfers between pages of memory are not supported. The page of graphics memory which will be affected by Raster operations is the current command page.

Buffering Procedures

The buffering procedure, DrawScreenBuffer, is designed to enhance the speed of the Graphics Manager for those applications requiring maximum performance levels. It achieves this by eliminating the overhead associated with the BTOS Request mechanism.

Normally, each call to a Graphics Manager routine generates a BTOS Request( ) call (see the BTOS Operating System manual for details on Request) to send the Manager a request block, the contents of which the Manager uses in determining what action needs to be performed. The construction of this request block and its routing to and decoding by the Manager entail a certain fixed overhead.

This overhead is usually acceptable. However, when a program consists of many small requests to the Graphics Manager, i.e. draw many small lines, this overhead becomes larger as a percentage of overall processing time, and throughput degrades somewhat.

Thus, for applications making such a sequence of requests, along with those which require improved performance for other reasons, a method of circumventing this overhead is provided.

The method used is simple: instead of passing each command via a request, a buffer of commands is passed to the Manager. Passing this buffer requires only one Request, as opposed to the number of commands in the buffer if each command stored in the buffer were requested individually.

DrawScreenBuffer

DrawScreenBuffer accepts a buffer of Graphics Manager commands, decodes the buffer, and performs the indicated requests. By reducing to 1 the number of BTOS Request() calls needed, this results in a faster execution time for the sequence of commands.

The buffer format used is as follows. An individual command is described by its request code, followed by the arguments to the requested routine in the order they are given in that routine's procedural interface. A buffer of commands is simply a sequence of individual commands.

The address of this buffer is DrawScreenBuffer's first argument. The other two are the size of the buffer (in bytes), and a pointer to a 4-byte error area. If an error is detected during the execution of the commands in the buffer, DrawScreenBuffer returns the error code from the command which caused the error. Further, a pointer to the part of the buffer which was being interpreted when the error occurred is written to the error area. The word pointed at by this pointer will be the request code of the failing routine, unless the error is one of those internal to DrawScreenBuffer.

The maximum size of a buffer is 64K. Further, the entire buffer must be addressable using the segment portion of the passed buffer address, or errors will result. A buffer address with an offset portion of 0 is guaranteed to meet this requirement.

Note: *Any Graphics Manager request may be buffered (including DrawScreenBuffer itself). The initial InitScreenGraphics() call, however, may not be buffered, since the DrawScreenBuffer call will not be accepted unless graphics is already initialized.*

Procedural Interface

DrawScreenBuffer (*pBuffer,cbBuffer,pError*): *ErcType*

pBuffer: Pointer to a buffer of commands.
cbBuffer: Size of the buffer in bytes.
pError: Pointer to 4-byte error address field.

ErcType:

0: No Error
7601: Graphics Not Initialized
7805: Buffer Size Count Invalid
7846: Invalid Request Code in Buffer
Additionally, DrawScreenBuffer may return any of the error codes which may be returned by any of the buffered GM commands.

Request Block

Offset	Field	Size (bytes)	Contents
0	sCntInfo	1	6
1	RtCode	1	0
2	nReqPbCb	1	1
3	nRespPbCb	1	1
4	userNum	2	
6	exchResp	2	
8	ercRet	2	
10	rqCode	2	0C07Dh
12	reserved	6	
18	pBuffer	4	
22	cbBuffer	2	
24	pError	4	
28	reserved	2	

Graphics Library Procedures

Introduction

This section contains detailed information about the Graphics Library. It contains a typical Graphics Library sequence and specific information about each Graphics Library procedure. For a general information about Graphics Library, see Section 1, the discussion of Graphics Library procedures found in Section 4, and Appendix G. Appendix A contains a sample Graphics Library program.

A Typical Graphics Library Sequence

The following steps are presented as a guideline to illustrate typical usage of the Graphics Library procedures. In this example, a picture is opened, an object is created, and the picture is saved in a picture file. The procedures used to accomplish each step are included in parentheses.

1 Allocate memory for the picture file work area (AllocMemorySL).

2 Initialize the graphics system (InitGraphics).

3 Set Column to 80 or 132 on B 27 workstations (SetColumnMode).

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

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

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

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

8 Close the object (CloseObject).

9 Save the picture (ClosePicture).

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

The Procedures

The Graphics Library procedures are organized in this section according to their general function. The order in which you will want to use procedures from the different groups depends on what you want your application to do. The arrangement used in this guide follows a logical top-down graphics processing sequence. The groups of procedures are shown in Table 8-1.

This section contains a subsection for each group of Graphics Library 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 Section 5.

A user-replaceable 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.

Table 8-1 **Graphics Library Procedures by Function**

1 Initialization

ClearViewport
InitGraphics
SetColumnMode
SetLimits
SetOutputDevice
SetOutputType
SetPlotterDevice
SetPlotterMaterial
SetUpGraphicsSpooling
SetUserCoordinates

2 Picture

AddPicture
ClosePicture
DisplayPicture
GetNumberOfObjects
OpenPicture
WritePicture

3 Object

AddObject
ClearLabels
ClearVectors
CloseObject
CloseTempObject
DisplayCurrentObject
OpenTempObject
RemoveCurrentObject
SetFirstObject
SetNextObject

4 Attribute

GetPictureColors
SetColor
SetCurrentPalette
SetDrawingMode
SetLineType

5 Drawing

Draw
DrawArc
DrawCircle
DrawLine
DrawRelative
FillRectangle
Move
MoveRelative

6 Text

SetCharacterSize
SetFont
SetLabelOrigin
WriteTextString

7 Font

GetFontName
GetFontNumber
GetNumberOfFonts
GetUserFontName
SetUserFont

8 Label

AddLabel
DeleteCurrentLabel
GetCurrentLabel
GetLabelData
ModifyLabel
SetFirstLabel
SetNextLabel

9 Transformation

GetTransformationData
SetScale
SetScaleRelative
SetTranslate
SetTranslateRelative

10 Viewing

GetWindowData
SetViewport
SetWindow

11 Cursor

GetCursorPosition
SetNDCCursorPosition
SetObjectCursorPosition
SetWorld CursorPosition
TurnOffCursor
TurnOnCursor

12 User Replaceable Routines

LoadPaper
ReadInterruptKey
SetPen

Initialization Procedures

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

There are ten initialization procedures:

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

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

ClearViewport

ClearViewport clears the viewport.

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

Procedural Interface

ClearViewport: *ErcType*

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued

InitGraphics

You MUST call InitGraphics before you call any other graphics procedure.

InitGraphics initializes the variables used by Graphics Library. The current command screen on the graphics hardware is cleared, and the default line type and drawing mode values are set. The window and viewport values are initialized by this procedure. Their default values can be found in Appendix D.

Procedural Interface

InitGraphics: *ErcType*

ErcType:

0:	No Error
7602:	An internal graphics error has occurred
7691:	Font file does not exist

SetColumnMode

You will need to use this command only if you are using a B 27 monochrome workstation. This procedure doesn't work on a B 27 color workstation.

Sets the column mode to either 80 or 132 on the B 27 workstation for both the graphics and character screen. If you need to run an application on a B 27 monochrome workstation, you MUST issue this call after an InitGraphics call since there is no default value for the column mode on the B 27. If you do not use this call, the application will assume the column mode used by the previous task, which may give you unwanted results.

The viewport and window values will change to reflect the new column mode setting.

The alpha video character map is reset using only one frame if more than one frame is desired, issue the alpha video commands (BTOS vol 2) with the appropriate parameters.

Note: *You don't need to use this routine if changing column modes in the alpha video routines prior to the InitScreenGraphics command. The alpha video routines will do it for you.*

Procedural Interface

SetColumnMode (*iColumnMode*): *ErcType*

iColumnMode: specifies the column mode desired.

On the B 21, B 22, B 26, and B 28 workstations, only 80 columns are valid.

On the B 27 you can choose either 80 or 132 columns.

ErcType:

0:	No error
7646:	Incorrect parameter for iColumnMode

SetLimits

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. See Appendix D for the aspect ratios for world and NDC coordinates for each workstation.

Procedural Interface

SetLimits (*rXMin, rYMin, rXMax, rYMax*): *ErcType*

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.

ErcType:

0: No Error
7646: Bad parameters specified

SetOutputDevice

SetOutputDevice allows you to choose whether to send your image to a dot matric printer, or a plotter, instead of the video display screen, which is the default.

See the User-Replaceable Procedures subsection for information on application procedures that can be called by Graphics Library procedures to extend the capabilities for plotter output processing.

Note: *If your output device is something other than the video screen, it must be reset to the screen—via SetOutputDevice—before ClosePicture is called.*

Note: *If you are not setting the ouput device as the screen, you must call SetOutputType, SetPlotterDevice, and SetPlotterMaterial before SetOutputDevice.*

Procedural Interface

SetOutputDevice (*iDevice*): *ErcType*

iDevice: Specifies the output device.

0 = video display screen
1 = plotter
2 = dot matrix printer

ErcType:

0:	No Error
7602:	Internal graphics error
7690:	Bad printer specification

SetOutputType

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

For file output, iOutputType must contain one of the output devices listed below since the file will be stored as if it was going to that output device. This is used in conjunction with SetPlotterDevice .

Note: *To obtain circles and circular arcs in printer output, you must use the ratio* 19:16.5 *(horizontal to vertical) for the variables rXMax and rYMax in the procedures* AddObject *and* SetViewport *(via the procedures* DrawArc *and* DrawCircle. *See the AddObject procedure for additional information.*

Procedural Interface

SetOutputType (*iOutputType*): *ErcType*

iOutputType: Specifies the code for the output device.

The following printers/plotters are available through the Graphics Library and are supported by Burroughs:

5 = AP1351 and AP1351-1 (132 column mode only)
6 = B9253
13 = AP1311
14 = AP1314
= AP1354 (80 column mode only)
15 = AP1351 and AP1351-1 (80 column mode only)
16 = AP1354 (132 column mode only)
17 = AP9208 (portrait mode)
18 = AP9208L (landscape mode)

The following printers/plotters are available through the Graphics Library, but are NOT supported by Burroughs:

0 = HP7470A
1 = HP7220C
2 = Strobe100
3 = Printronix MVP
4 = Anadex 9620
7 = Envision 420
8 = not used
9 = HP7475A
10 = HP7220T
11 = Okidata Microline 93
12 = Data Products 8010

ErcType:

0:	No Error
7693:	Output device specified is not valid

SetPlotterDevice

SetPlotterDevice specifies one of the following for output devices:

- Name of a disk file
- Configuration file
- Queue name

When using devices through the COMM port, either [COMM]A or [COMM]B must be specified (depending on which port is used).

The following configurations are necessary for direct printing or plotting.

HP7470A and HP7475A

[COMM]A&[Sys]<sys>PlotterConfig.sys
[COMM]B&[Sys]<sys>PlotterConfig.sys

AP1311, AP1351, AP1351-1, AP1314, and AP1354

[LPT]&[Sys]<Sys>GraphicsPrinterConfig.sys

AP9208 and AP9208L

[PTR]B&LaserPrinterConfig.sys

B9253

For spooled printing or plotting, the application queue name is entered. For example, if the queue name for the B9253 is B9253Q, then enter [B9253Q]. See Section 10 for additional information on spooling to printers and plotters.

Note: *Note: To obtain circles and circular arcs in printer output, you must use the ratio* 19:16.5 *(horizontal to vertical) for the variables rXMax and rYMax in the procedures* AddObject *and* SetViewport *(via the procedures* DrawArc *and* DrawCircle*). See the AddObject procedure for additional information.*

Procedural Interface

SetPlotterDevice (*pbDevName,cbDevName*): *ErcType*

pbDevName, cbDevName: Describe the disk file name, queue name or the configuration file for the device.

ErcType

0: No Error

SetPlotterMaterial

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*

iMaterial: Specifies whether the output is to be on paper or on a transparency.

0 = paper
1 = transparency

ErcType:

0: No Error

SetUserCoordinates

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*

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.

ErcType:

0: No Error
7601: InitGraphics must be the first call issued
7602: An internal graphics error has occurred
7620: Object is not open
7646: Bad parameters were supplied

SetUpGraphicsSpooling

This procedure sets a flag to spool the output to either a printer, plotter or a file. See Table 8-2 for the proper configurations.

Note: *You must set the flag fSpool before spooling the output to a device or file.*

Procedural Interface

SetUpGraphicsSpooling (*fSpool, fSpoolToFile*): *ErcType*

fSpool: Set to TRUE to spool all output. (Defaults to FALSE, direct printing.)

fSpoolToFile: Set this flag to TRUE if the output is being spooled to a file rather than a device. (Defaults to FALSE; assumes printing to device, not file.)

ErcType:

0: No error

Table 8-2 **Spool Configurations**

Type of Output	fSpool	fSpooltoFile
Direct to Printer/Plotter	FALSE	FALSE
Spooled output to a printer	TRUE	FALSE
Spooled output to a plotter	TRUE	FALSE
Spooled output to a file	TRUE	TRUE

Picture Procedures

The picture procedures are used to manage picture files and to manipulate pictures on the current command screen. When a new graphic representation is being created, a picture must be 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 created, modified, and transformed. When the current picture has been completed, it should be written to a picture file and closed. After the current picture is closed, another picture can be processed.

There are six picture procedures:

- Addpicture
- ClosePicture
- DisplayPicture
- GetNumberOfObjects
- OpenPicture
- WritePicture

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

AddPicture

AddPicture adds the specified picture file to the current picture. The added picture becomes part of the current picture.

In color graphics workstation applications, the fOverwrite-Palette 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 (*pbPictureN ame, cbPictureName, fOverwritePalette*): *ErcType*

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.

0FFh = TRUE, overwrite
00h = FALSE, do not overwrite

ErcType:

0:	No error
7602:	An internal graphics error has occurred
7610:	Picture was not open
7620:	Object not open
7628:	Tried to acess an object past the last object
7642:	Font file was removed—it does not exist
7649:	Insufficient memory

ClosePicture

ClosePicture closes the current picture. If the parameter fSave is set to TRUE, the picture is saved in the picture file previously specified in the OpenPicture command before it is closed. If the fSave parameter is set to FALSE the picture is not saved.

Procedural Interface

ClosePicture (*fSave*): *ErcType*

fSave: Specifies whether the picture is to be written before it is closed.

0FFh = TRUE
00h = FALSE

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7610:	Picture not opened

DisplayPicture

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 and can overlay 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 zero status code which prompts DisplayPicture to continue writing the output without an interruption. ReadInterruptKey can be replaced by a user-replaceable procedure with the same name to halt the DisplayPicture Process. See User-Replaceable Procedures 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. See User-Replaceable Procedures for detailed information about the use of SetPen.

Note: *If your output device is something other than the video screen, you should reset it to the screen—via SetOutputDevice (0)—after each DisplayPicture.*

Procedural Interface

DisplayPicture (*flnterruptOnKey*): *ErcType*

flnterruptOnKey: indicates whether or not ReadInterruptKey is to be called.

0FFh = TRUE, ReadInterruptKey is called.
00h = FALSE, it is not.

ErcType:

0:	No Error
7642:	Font name specified does not exist.
7649:	Insufficient memory
7610:	Picture must first be opened before calling this procedure.

GetNumberOfObjects

GetNumberOfObjects returns the number of objects in the current picture.

Procedural Interface

GetNumberOfObjects (*pNObjectsRet*): *ErcType*

pNObjectsRet: Points to the memory address of a word where the number of objects in the picture is to be returned.

ErcType:

0: No Error

OpenPicture

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

One of the 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 other picture files to create a complex picture. Write mode can also be used to overwrite existing pictures from pictures 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 large enough to contain the whole picture. A simple picture requires approximately 16K, and a complex picture 48K.

Procedural Interface

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

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 (exclusive) or 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 Burroughs programming languages.

pMemory
cParasMemory: Specifies the segment of memory to be used as a workarea for the picture. The memory size indicates the number of 16-byte sections allocated.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7611:	Picture is already open
7613:	Unauthorized password
7614:	Picture file specified does not exist
7621:	Object is currently open
7649:	Insufficient memory

WritePicture

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*

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

ErcType:

0: No Error
7610: Picture not opened

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 which are used to select the current object in a picture. 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

AddObject is used to begin a new object that is to be part of the current picture. The object specified by pbObjectName 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. Only one object can be open at any given time. To obtain a circle or circular arc on the screen, use the aspect ratios in Appendix D.

To obtain circles and circular arcs on the screen as well as in printer output, you must use the ratio *19:16.5* (horizontal to vertical) for the variables rXMax and rYMax. This is required in *SetViewport* as well as in *AddObject* (via the procedures *DrawArc* and *DrawCircle*).

Procedural Interface

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

pbObjectName *cbObjectName*:	Specifies the name of the object 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.

ErcType:

0:	No Error
7602:	An internal graphics error was encountered
7610:	Picture not open
7612:	Picture is Read Only
7621:	Object is already open
7622:	A bad object name was specified
7646:	A bad parameter was specified
7649:	Insufficient memory

ClearLabels

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*

ErcType:

0:	No Error
7610:	Picture is not open
7612:	Picture is Read Only
7620:	Object is not open

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. There is no change to the graphics screen.

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*

ErcType:

0:	No Error
7610:	Picture is not open
7612:	Picture is Read Only
7620:	Object is not open

CloseObject

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*

ErcType:

0:	No Error
7610:	Picture is not open
7620:	Object is not open
7625:	The object specified is a temporary one

CloseTempObject

CloseTempObject closes a temporary object.

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

Procedural Interface

CloseTempObject: *ErcType*

ErcType:

0:	No Error
7620:	Object is not open
7624:	Object specified is not temporary

DisplayCurrentObject

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*

ErcType:

0:	No Error
7610:	Picture must first be open
7620:	Object must first be open.

OpenTempObject

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*

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.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7621:	Object is already open
7629:	Temporary objects cannot be opened when a picture is open.
7646:	Bad parameters were specified

RemoveCurrentObject

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*

ErcType:

0:	No Error
7610:	Picture is not open
7612:	Picture is Read Only
7620:	Object is not open

SetFirstObject

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*

ErcType:

0:	No Error
7620:	Object is not open
7628:	Tried to access an object past the last object

SetNextObject

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, an error code is returned:

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

A picture must be open before SetNextObject can be used.

Procedural Interface

SetNextObject: *ErcType*

ErcType:

0:	No Error
7628:	Tried to access an object past the last object
7620:	Object is not open

Attribute Procedures

Attribute procedures are used to set the values for attributes that are used in conjuction with drawing procedures. The attributes are line type, drawing mode, and color. Detailed information about these drawing attributes is included in Section 4, Concepts.

Before an attribute procedure can be called, 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 opened.

There are five attribute procedures:

- GetPictureColors
- SetColor
- SetCurrentPalette
- SetDrawing Mode
- SetLineType

GetPictureColors

GetPictureColors returns 8 bytes that define the colors in the current palette. The color bytes are copied to the memory location specified by the parameter pRgbPaletteRet. See Graphics Manager Requests for detailed information about color palettes.

Procedural Interface

GetPictureColors (*pRgbPaletteRet*): *ErcType*

pRgbPaletteRet: Specifies the memory address of the buffer where the n bytes used to define the current palette are to be returned.

ErcType:

0: No Error
7610: Picture is not open

SetColor

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 n-byte color palette. The acceptable values are 1 through 8. Multi-color output to a video display unit is supported only on the color graphics workstations.

When a monochrome 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 Section 4.

Procedural Interface

SetColor (*iColor*): *ErcType*

iColor: Specifies the color to be used in subsequent commands, in the range 1–8. The default is 1.

ErcType:

0: No Error
7601: Graphics Not Initialized.
7620: Object must first be open.
7627: Object was not opened in write mode.
7693: Device specified must either be the screen or a plotter, not the printer.

SetCurrentPalette

SetCurrentPalette is used in application systems designed for color graphics workstations. It specifies the n bytes that define a new palette. The palette that is selected remains the current palette for subsequent drawing procedures until SetCurrentPalette is called again. For detailed information about the use of color palettes, see Section 4.

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*

pRgbPalette: Points to the memory address of the 8 bytes that define the colors in the palette.

ErcType:

0: No Error

SetDrawingMode

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 iDrawingMode. Detailed information about the drawing modes, including an illustration, can be found in the subsection Drawing Attributes in Section 4.

The default is 0. (SetMode)

Procedural Interface

SetDrawingMode (*iDrawingMode*): *ErcType*

iDrawingMode: Specifies the drawing mode.

0 = Set Mode
1 = Clear Mode
2 = Complement Mode
3 = Replace Mode

ErcType:

0:	No Error
7646:	Drawing mode specified is not valid
7649:	Insufficient memory

SetLineType

SetLineType specifies the current line type. 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 section.

Procedural Interface

SetLineType (*iLineType*): *ErcType*

iLineType: Specifies the line type to be used in subsequent drawing procedures.

0–7 = The standard line types. See Figure 4-1.

ErcType:

0: No Error
7646: Line type specified is invalid
7649: Insufficient memory

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. You must call drawing procedures prior to setting the output device if it is to be a printer.

There are eight drawing procedures:

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

Draw

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*

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.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7620:	Object is not open
7627:	Object was not opened in Write Mode
7649:	Insufficient memory
7693:	Output device is not a valid one

DrawArc

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 7-2 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.

Note: *DrawArc will actually draw an elliptical curve instead of a round one. If you need to create the appearance of a round curve, set up the AddObject command using the aspect ratio of the hardware system the application will run on. See Appendix D to find the correct aspect ratio for your workstation.*

Procedural Interface

DrawArc (*rXCenter, rYCenter, rsRadius, rAngl, rAng2*): *ErcType*

rXCenter *rYCenter*:	Specify the position from which the angles are calculated to create the arc.
rsRadius:	Specifies the radius from the center point.
rAngl:	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.

ErcType:

0:	No Error
7601:	InitGraphics must be thefirst call issued
7602:	An internal graphics error occurred
7620:	Object is not open.
7627:	Object was not opened inWrite Mode
7693:	Output device is not a valid one.

DrawCircle

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.

Note: *The DrawCircle command actually draws an elliptical curve rather than a round curve. If you need to create the appearance of rounded curves, set up the AddObject command with the aspect ratio of the hardware system the application will run on. See Appendix D to find the correct aspect ratio for your workstation.*

Procedural Interface

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

rXCenter
rYCenter: Specify the center of the circle.

rSRadius: Specifies the radius of the circle.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7620:	Object is not open
7627:	Object was not opened inWrite Mode
7693:	Output device is not a valid one

DrawLine

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*

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.

ErcType:

0: No Error
7601: InitGraphics must be the first call issued
7602: An internal graphics error occurred
7620: Object must first be opened
7649: Insufficient memory

DrawRelative

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*

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.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7620:	Object must first be opened
7627:	Object was not opened in Write Mode
7649:	Insufficient memory
7693:	Output device is not a valid one

FillRectangle

FillRectangle fills a rectangle with a pattern. There are six different patterns. Appendix B contains an illustration of these fill patterns.

Procedural Interface

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

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.

Appendix B contains an illustration of the fill patterns available.

ErcType:

0: No Error
7601: InitGraphics must be the first call issued
7602: An internal graphics error occurred
7603: RasterOps are not available
7610: Picture must first be opened
7648: Parameters outside the viewport were specified
7649: Insufficient memory

Move

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*

rX: Specifies the X coordinate for the new current position.

rY: Specifies the Y coordinate for the new current position.

ErcType:

0:	No Error
7601:	Graphics not initialized.
7620:	Object must first be opened.
7627:	Object was not opened in write mode.
7646:	Parameters given are outside of specified viewport.
7693:	An invalid device was specified

MoveRelative

MoveRelative sets the new current position to the existing current position with an offset of 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*

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.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7620:	Object must first be opened
7627:	Object was not opened inWrite Mode
7649:	Insufficient memory
7693:	

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 only, 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, see the Text Attributes in this section.

SetCharacterSize

SetCharacterSize specifies the relative character size that will be used in subsequent WriteTextString procedures. See Text Attributes for detailed information about the character size attribute.

Procedural Interface

SetCharacterSize (*rsChars*): *ErcType*

rsChars: Specifies the character size used in subsequent calls to WriteTextString.
1 = Standard (the default)

ErcType:

0:	No Error
7620:	Object must first be opened
7627:	Object was not opened inWrite Mode
7649:	Insufficient memory
7693:	Invalid Output Device.

SetFont

SetFont specifies the font that will be used in subsequent WriteTextString procedures. Graphics for BTOS Systems contains five fonts:

- SimplexRoman (the default font),
- ComplexRoman,
- SimplexPlot
- DuplexRoman and
- Gothic.

See the Text Attributes for detailed information about the font attribute. Procedures for selecting fonts and establishing which fonts are available are found in the discussion of Font Procedures.

Procedural Interface

SetFont (*pbFontName, cbFontName*): *ErcType*

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

ErcType:

0:	No Error
7642:	Invalid font name was given
7649:	Insufficient memory

SetLabelOrigin

SetLabelOrigin specifies the current label origin to be used in subsequent WriteTextString procedures. Figure 4-3 illustrates this concept. 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. See Text Attributes in this section for detailed information about the label origin attribute.

Procedural Interface

SetLabelOrigin (*bLorg*): *ErcType*

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

ErcType:

0:	No Error
7620:	Object must first be opened
7627:	Object was not opened in Write Mode
7643:	Invalid Label Origin was given
7649:	Insufficient memory
7693:	Output device is not a valid one

WriteTextString

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. See the Text Attributes subsection for detailed information 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*

pbString
cbString: Describe the string to be drawn at the current position.

ErcType:

0:	No error
7602:	An internal graphics error occurred
7620:	Object must first be opened
7626:	Object is Read Only
7627:	Object was not opened in write mode
7644:	Character is out of bounds
7649:	Insufficient memory
7693:	Output device is not a valid one

Font Procedures

The graphics software contains five fonts. The name for each font that is used internally by the graphics software is called the *internalname*. 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

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 five fonts that are included with the graphics software are:

SimplexRoman
ComplexRoman
SimplexPlot
DuplexRoman
Gothic

file specification: Specifies the volume, directory, and file name of the font.

There are five font procedures:

- GetFontName
- GetFontNumber
- GetNumberOfFonts
- GetUserFontName
- 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.

Table 8-3 shows the Graphics.Fonts entries supplied with the graphics software.

Table 8-3 **Graphics.Fonts File Names**

Standard: SimplexRoman:

Complex:	ComplexRoman: [SYS]<SYS>ComplexRoman.font
Simplex:	[SYS]<SYS>SimplexPlot.font
Bold:	DuplexRoman: [SYS]<SYS>DuplexRoman.font
Gothic:	Gothic: [SYS]<SYS>Gothic.font

Note: *These are example file names. Your font files may reside in different volumes and/or directories.*

GetFontName

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*

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.

ErcType:

0: No Error
7691: Number given for ifont is out of range of number of fonts in file Graphics.Fonts

GetFontNumber

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*

pbFontName *cbFontName*: Specify an internal font name.

piFontRet: Points to the memory location where the font number is to be returned.

ErcType:

0: No Error
7642: Font name given does not exist

GetNumberOfFonts

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*

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

ErcType:

0: No Error

GetUserFontName

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*

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.

ErcType:

0:	No Error
7691:	Font number does not exist

SetUserFont

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*

pbFontName Specify the user-friendly name of the font.
cbFontName:

ErcType:

0:	No Error
7620:	Object is not open
7627:	Object was not open in write mode
7642:	Font name does not exist
7649:	Insufficient memory
7693:	Output device is not a valid one

Label Procedures

The label procedures are used to create labels to accompany the vector portion of an object. Once you create a label, it is part of the picture, since it is created using the graphics capablilities. 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.

Graphics for BTOS Systems uses vector fonts to create alphabetic letters and numerals. To use alphanumeric text, instead of graphics labels, see *BTOS Reference Manual Volumes 1 and 2* for details.

The label structure is used when modifying an existing label. The current label is moved into this workarea structure and the modifications are made in the workarea. After all modifications are completed, the workarea is then automatically copied back into the object to replace the current label. The coordinate positions, text, and attributes of the label are saved in the label list.

Appendix H, Label Structures, shows the format of the label structure.

There are seven label procedures:

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

AddLabel

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 specified position in the world coordinate system. (See Figure 4-3.) 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 five 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 subsection Text Attributes. 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 before this command is executed.

Procedural Interface

AddLabel (*rX, rY, pbString, cbString, rsChars, bLorg, bPen, bUserID, pbFontName, cbFontName*): *ErcType*

rX *rY*:	Specifies the position of the label
pbString *cbString*:	Specifies 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 color graphics workstation applications, the color number

bUserID: Specifies the label identifier.

pbFontName
cbFontName: Specifies the internal name of the font to be used for the label.

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7610:	Picture must first be open
7612:	Picture is Read Only
7620:	Object must first be open
7642:	Font name given does not exist
7644:	Character is out of bounds
7645:	No characters were given for the label (cbString = 0)
7649:	Insufficient memory
7693:	Output device is not a valid one

DeleteCurrentLabel

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 DeleteCurrentLabel is used. The output cannot be a printer when this command is executed.

Procedural Interface

DeleteCurrentLabel: *ErcType*

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7610:	Picture is not open
7612:	Picture is Read Only
7620:	Object must first be opened
7640:	GetCurrentLabel must first be called
7693:	Output device is not a valid one

GetCurrentLabel

GetCurrentLabel copies the current label from the current object's label list into a workarea structure where it can then be modified. In order to access the entire data structure, 66 bytes (plus the length of the label in bytes) of memory, must be allocated for the workarea. See Appendix H, Label Structures for more information about the label structure. A label must be designated as the current label before GetCurrentLabel is used.

Procedural Interface

GetCurrentLabel (*pLabelRet, sLabelRet*): *ErcType*

pLabelRet: Points to a structure where the label is to be copied.

sLabelRet: Specifies the maximum size of the structure.

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7610:	Picture must first be opened
7612:	Picture is Read Only
7620:	Object must first be opened
7640:	GetCurrentLabel must first be called
7649:	Insufficient memory

GetLabelData

GetLabelData computes and fills in the boundaries of a label located in a workarea structure. See the introduction to this subsection, Label Procedures for information about the label structure, including which parameters have values returned by GetLabelData. Also see Appendix H, Label Structures.

This procedure is used to determine if a new label will fit, as is, in the world coordinate system. If it does not fit, an ercCharOutOfBounds (error 7644) message is returned.

Procedural Interface

GetLabelData (*pLabelRet*): *ErcType*

pLabelRet: Points to the structure containing the label.

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7642:	Font name is not a valid one
7644:	Character is out of bounds
7649:	Insufficient memory

ModifyLabel

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. See 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*

pModifiedLabel: Points to the structure that contains the modified label.

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7610:	Picture must first be open
7612:	Picture is Read Only
7620:	Object must first be open
7640:	GetCurrentLabel must first be called
7642:	Font name does not exist

7644: Character is out of bounds
7645: No characters were given for the label (cbString = 0)
7649: Insufficient memory
7693: Output device is not a valid one

SetFirstLabel

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. See the introduction to this subsection, Label Procedures, for information about the label structure.

Procedural Interface

SetFirstLabel (*pLabelRet, sLabelRet*): *ErcType*

pLabelRet: Points to a structure where the label is to be copied.

sLabelRet: Specifies the maximum size of the structure.

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7610:	Picture must first be open
7612:	Picture is Read Only
7620:	Object must first be open
7641:	End of Label List was encountered
7649:	Insufficient memory

SetNextLabel

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, an ercEndOfLabelList error message is returned, since there is no next label.

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. See the introduction to this subsection, Label Procedures, for information about the label structure.

Procedural Interface

SetNextLabel (*pLabelRet, sLabelRet*): *ErcType*

pLabelRet: Points to a structure where the label is to be copied.

sLabelRet: Specifies the maximum size of the structure.

ErcType:

0:	No Error
7602:	An internal graphics error occurred
7610:	Picture must first be open
7612:	Picture is Read Only
7620:	Object must first be open
7641:	End of Label List was encountered
7649:	Insufficient memory

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. 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. Rotation is not supported on this product.

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

There are five transformation procedures:

- GetTransformationData
- SetScale
- SetScaleRelative
- SetTranslate
- SetTranslateRelative

GetTransformationData

GetTransformationData returns the transformation values for the current object. The values are returned in the following data structure of the variable pTransformRet:

Offset	Field	Size(bytes)
0	rXScale	4
3	rXTranslate	4
7	rYScale	4
11	rYTranslate	4

Procedural Interface

GetTransformationData (*pTransformRet*): *ErcType*

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

ErcType:

0: No Error
7601: InitGraphics must be the first call issued

SetScale

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 scaling causes any part of the object to be moved outside the world coordinate area, the error message ercBadTransformationParameter will be returned.

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*

rXScale: Specifies the scaling of the object in the X direction.

rYScale: Specifies the scaling of the object in the Y direction.

ErcType:

0: No Error
7601: InitGraphics must be the first call issued
7602: An internal graphics error
7620: Object must first be open
7647: Scale parameter too big

SetScaleRelative

SetScaleRelative is used after SetScale to scale the object further 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, an erc BadTransformationParameter error message is returned:

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

Procedural Interface

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

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.

ErcType:

0: No Error
7601: InitGraphics must be the first call issued
7602: An internal graphics error occurred
7620: Object must first be open
7647: Scale parameter too big

SetTranslate

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, an ercBadTransformationParameter error message is returned.

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*

rXTranslate: Specifies the translationin the X direction.

rYTranslate: Specifies the translation in the Y direction.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error occurred
7647:	Translate parameter is too big
7620:	Object must first be open

SetTranslateRelative

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

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

Procedural Interface

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

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.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued
7602:	An internal graphics error
7620:	Object must first be open
7647:	Translate parameter is too big

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 pictures are unchanged.

There are three viewing procedures:

- GetWindowData
- SetViewport
- SetWindow

GetWindowData

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

Procedural Interface

GetWindowData (*pWindowData*): *ErcType*

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

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued

SetViewPort

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. See Appendix D.

Procedural Interface

SetViewPort (*rXMin, rYMin, rXMax, rYMax*): *ErcType*

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.

ErcType:

0:	No Error
7601:	InitGraphics must be the first call issued.
7602:	An internal graphics error occurred
7646:	Parameters specified are not valid. They are not contained within your workstation's defined viewport. See Appendix D.

SetWindow

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. See Appendix D.

Procedural Interface

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

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.

ErcType:

0: No Error
7602: An internal graphics error occurred
7646: An invalid parameter was given

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. See Appendix D.

There are six cursor procedures:

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

GetCursorPosition

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

Procedural Interface

GetCursorPosition (*pCursorStatusRet*): *ErcType*

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)

ErcType:

0:	No Error
7602:	An internal graphics error occurred

SetNDCCursorPosition

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. See Appendix D, World and NDC coordinates.

Procedural Interface

SetNDCCursorPosition (*rX, rY, bDir*): *ErcType*

rX: Specifies the X coordinate for the cursor position.

rY: Specifies the Y coordinate for the cursor position.

bDir: Specifies the direction of the cursor arrow.

0 = up
1 = down
2 = right
3 = left

ErcType:

0: No Error
7646: An invalid parameter was given

SetObjectCursorPosition

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*

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

ErcType:

0: No Error
7602: An internal graphics error occurred

SetWorldCursorPosition

SetWorldCursor 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. See Appendix D.

Procedural Interface

SetWorldCursorPosition (*rX, rY, bDir*): *ErcType*

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

ErcType:

0: No Error
7602: An internal graphics error occurred

TurnOffCursor

TurnOffCursor turns off the cursor.

Procedural Interface

TurnOffCursor:*ErcType*

ErcType:

0: No error

TurnOnCursor

TurnOnCursor displays the cursor at its current location.

Procedural Interface

TurnOnCursor: *ErcType*

ErcType:

0: No error

User-Replaceable Procedures

There are three User-Replaceable procedures contained within the Graphics Library. These procedures are used independently of one another to perform separate tasks. They allow applicatioon programs to include special processing capabilities within the Graphics Library.

Calls to these procedures are made from various Graphics Library procedures which are discussed further in each of the User-Replaceable procedures. The user can replace any of these procedures with user written code to accomplish a variety of functions. To replace the existing code with your own code, you must remove the existing object code from the Graphics.Lib file and add your own object code to this file via the Librarian command. For instance, you may want to write the message from the LoadPaper procedure to a different location on your screen. If so, you could write new code for LoadPaper to do this. The global variables you need to manipulate these replaceable procerdures are explained in detail under each procedure. There are three Graphics Library procedures that can be replaced with user-replaceable versions:

- LoadPaper
- ReadInterruptKey
- SetPen

LoadPaper

LoadPaper displays a message on the screen to tell the user to load a piece of paper or transparency in the plotter. This procedure is called by SetOutputDevice.

Procedural Interface

LoadPaper (*fPaper*) : *ErcType*

fPaper: Indicates whether the output is to be paper or a transparency.

OFFh = TRUE, the output is paper.

00h = FALSE, the output is transparency.

ErcType:

0: No error

ReadInterruptKey

ReadInterruptKey enables an application to interrupt the process of displaying a picture. This procedure is called from within DisplayPicture if the parameter fInterruptOnKey is set to TRUE. After the interrupt occurs, processing continues within the DiplayPicture procedure. The default version of ReadInterruptKey returns a status code of zero.

Procedural Interface

ReadInterruptKey: *ErcType*

ErcType:

0: No error

SetPen

SetPen enables an application to halt the plotter output while the user changes one of the pens. A message on the video display screen instructs the users to switch pens. 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 halts, and the user is 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. SetPen returns the pen that is to be changed to its holder 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*

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

Configuring Application Programs for Graphics

Introduction

Before you can use your program with Graphics, you must reconfigure the language in which it is written. The following instructions, organized according to language, describe the steps you must take to do this. Refer to the appropriate language manuals if you need more detailed information about this process.

Applications written for B 20 Systems Graphics are source code compatible with BTOS Systems Graphics, but are not object code compatible. If you wish to run previously written applications on a B 27, you must use the procedure SetColumnMode. For all workstations, you must recompile and/or relink your source code with Graphics. Follow the steps below.

BASIC

The language needs to be reconfigured before graphics applications will work. This involves the following:

1 Edit the file <Sys>BasGen.Asm.

2 Remove the comment symbols that are next to the Table Entries (calls) in the Graphics section(s) that are being used in your application program(s).

3 Save the file when done.

4 Assemble the file <Sys>BasGen.Asm using the following instructions:

If you are using Graphics Manager calls or Graphics Library calls, always type **Yes** to following question:

Are you calling Graphics Manager?

Always type **Yes** after each of the following questions if you are calling the Graphics Library:

Are you calling ConvertTo8087?
Are you calling Graphics Manager?
Are you calling Graphics Library?

For both Graphics Manager and Graphics Library, Type **Yes** after all appropriate questions.

5 Submit the file <Sys>LinkBasic.Sub (for BASIC Interpreter only).

Note: *If you are calling Graphics Library routines, the following files must be specified in the linker form:*

Field	File
Object module	<Sys>gfxGPAMDum.obj
	<Sys>MapGraphicsWindow.obj
[Libraries]	<Sys>Graphics.lib

Press GO after filling in the appropriate information.

You **must** recompile all application programs that will be using graphics. Even if you are only using the Graphics Manager routines, the programs must be recompiled.

6 (*For BASIC Compiler only*) Recompile your program, including in the [BasGen File] field of the compiler form, <Sys>BasGen.obj.

To use the Basic Compiler, relink your programs after successfully compiling them. If you are using calls to the Graphics Library, be sure to include <sys>Graphics.lib in the [Libraries] field and<sys>gfxGPAMDum.obj in the Object Module field of the Linker form. Go to Step 4 if you are using the BASIC Interpreter.

Be sure to install Graphics Manager before you run a compiled BASIC program or use the BASIC interpreter.

COBOL

You must reconfigure COBOL in order to run graphics applications. Take the following steps:

1 Edit the file <Sys>CobolGen.Asm.

2 Remove the comment symbols that are next to the Table Entries (calls) in the Graphics section(s) that are being used in your application program(s).

3 Save the file when done.

4 Assemble the file <Sys>CobolGen.Asm using the following instructions:

Answer **YES** to "Are you calling Graphics Manager?" Always answer **YES** to this.

Answer **YES** to all other appropriate calls that your application program(s) will be using.

5 Submit the file LinkCobol.sub.

6 Press GO after filling in the appropriate information.

You **must** recompile all application programs that will be using graphics. Even if you are only using the Graphics Manager routines, the programs must be recompiled.

If the Graphics Manager has not been installed on your system, invoke the Install Graphics Manager command and press GO. You are now ready to use Graphics.

FORTRAN

The language needs to be reconfigured before graphics applications will work. This involves the following:

1 Edit the file <Sys>ForGen.Asm.

2 Remove the comment symbols that are next to the Table Entries (calls) in the Graphics section(s) that are being used in your application program(s).

3 Save the file when done.

4 Assemble the file <Sys>ForGen.Asm using the following instructions:

Answer **YES** to "Are you calling Graphics Manager?" Never answer **NO** to this.

Answer **YES** to "Are you calling Graphics Library?" if you are going to use these calls.

Answer **YES** to all other appropriate sections that your application program(s) will be using.

You **must** recompile all application programs that will be using graphics. Even if you are only using the Graphics Manager routines, the programs must be recompiled.

Relink the programs after successfully compiling them. Include the file <Sys>ForGen.obj in the "Object Modules" field of the Linker form.

Note: *If you are calling Graphics Library routines, the following files must be specified in the linker form:*

Field	**File**
Object module	<Sys>gfxGPAMDum.obj
	<Sys>MapGraphicsWindow.obj
[Libraries]	<Sys>Graphics.lib

If the Graphics Manager has not been installed on your system, invoke the Install Graphics Manager command and press GO. You are now ready to use Graphics.

Pascal

You must recompile all application programs that will be using graphics. Even if you are only using the Graphics Manager routines, the programs must be recompiled.

Relink the programs after successfully compiling them.

Note: *If you are calling Graphics Library routines, the following files must be specified in the linker form:*

Field	File
Object module	<Sys>gfxGPAMDum.obj
[Libraries]	<Sys>Graphics.lib

If the Graphics Manager has not been installed on your system, invoke the Install Graphics Manager command and press GO. You are now ready to use Graphics for BTOS Systems.

Printers and Plotters

This section contains information on printer and plotter configurations for all BTOS workstations. It explains how to create and modify configuration files for both direct and spooled printing and plotting. Read the section on printing in the *BTOS Standard Software Operations Guide* for more information on configuration files, direct printing and plotting, and spooled printing and plotting.

The Graphics Library installation disks contain six configuration files for printing and plotting, which are copied to your [Sys]<Sys> directory during installation.

The two Burroughs supported printer configuration files are:

GraphicsPrinterConfig.sys
LaserPrinterConfig.sys

The four nonsupported plotter configuration files are:

PlotterConfig.sys	Direct for Hewlett-Packard
HPPlotterConfig.sys	Spooled for Hewlett-Packard
StrobeConfig.sys	Direct for Strobe
StrobePlotterConfig.sys	Spooled for Strobe

Sixteen peripherals are used by the Graphics Library. Peripherals supported by Burroughs are:

Burroughs AP1311 Multi Function Printer
Burroughs AP1351 Multi Function Printer
Burroughs AP1351/1 Multi Function Printer
Burroughs AP1314 Dot Matrix Printer
Burroughs AP1354 Dot Matrix Printer
Burroughs AP9208 Non-Impact Laserc Printer (serial only)
Burroughs B9253 Dot Matrix Printer

Peripherals *not* supported by Burroughs are:

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

Direct and Spooled Printing

In *direct printing*, a printer or plotter is physically attached to one workstation and it cannot be shared by any other workstations on the cluster. During direct printing, the printer will not process requests unless it is in a Ready state and no other requests can be processed until printing is complete. In *spooled printing*, a printer or plotter is shared by the other workstations on the cluster, regardless of where it is physically attached. During spooled printing, print requests are sent to a queue, which allows the processor to continue with the next task. Spooled peripherals must be recognized by the Queue Manager.

Installing Printers and Plotters

To install a printer/plotter on your system for spooling, complete the following steps:

Note: *If you are spooling to a AP9208 laser printer, and your BTOS workstation is connected to an XE 520, do the following:*

- ☐ Connect the AP9208 laser printer to one of the RS 232 ports on your workstation.
- ☐ Install the spooler at your workstation.

1. Verify that your printer/plotter switch settings match the parameters in your configuration file.
2. Create/Modify the Queue.Index file.
3. Create/Modify a Spooler configuration file (SplCnfg.sys).
4. Reboot your system if you create or modify either the Queue.Index or SplCnfg.sys files.
5. Install the Queue Manager.
6. Install the spooler.

Configuration File Contents

The following printer and plotter configuration files, labeled A through F, correspond to configuration files A through F in Table 10-1. Look up your printer or plotter configuration file in Table 10-1 and use the corresponding configuration file parameters to set your printer or plotter switches for Graphics. See your printer or plotter operator's guide for more information on setting switches.

A	PlotterConfig.sys	Hewlett-Packard (direct)
B	HPPlotterConfig.sys	Hewlett-Packard (spooled)
C	StrobeConfig.sys	Strobe (direct)
D	StrobePlotterConfig.sys	Strobe (spooled)
E	GraphicsPrinterConfig.sys	Printing (direct and spooled)
F	LaserPrinterConfig.sys	AP9208 (direct and spooled, serial only)

Table 10-1 **Printer and Plotter Switch Settings**

Configuration File Parameters	**Configuration Files**		
	A	**B**	**C**
Data bits	7	7	8
Parity	0	0	
Baud rate	2400	2400	2400
Stop bits	1	1	1
Transmit time out	60	60	5
Receive time out	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		132	

Configuration File Parameters	Configuration Files		
	D	E	F
Data bits	8		8
Baud rate	2400		9600
Stop bits	1		1
Transmit time out	60	60	20
Newline mapping mode	binary	binary	binary
Line control	XonXoff		XonXoff
Tab expansion size	8	0	0
Number of characters per line	80	132	132
Additional ACK delay		0	

Note: *Blank spaces in the table indicate that the associated parameter is not applicable.*

Your release disk contains three *Sample* files to help you set up the appropriate spooler, queue, and printer configurations for your system. These files are:

Sample>QueueIndex

Sample>SplCnfg.Sys

Sample>Sys.Printers

Do not overwrite your corresponding <Sys> files with any of these samples. Instead:

1 Examine the contents of each sample file.

2 Identify the individual configuration lines appropriate for your system.

3 Copy the appropriate lines from the sample file into your existing configuration file.

Queue.Index File

The file Queue.Index resides on the master workstation. You can modify this file to accept new queue names for spooled output. If it does not exist on the master, you can create this file.

To modify an existing Queue.Index file, enter the necessary fields as follows:

QueueName/FileSpec/EntrySize/QueueType

QueueName A user-defined name that is unique to the installation and less than 50 characters in length with no spaces. These names must not be the default system device names.

FileSpec This is the queue entry file that stores queue entries. Use the [Sys]<Spl> directory for this file.

EntrySize Specifies the size of an entry for the queue entry file in 512-byte units. The standard entry size is *1*.

QueueType Specifies the type of queue. This number must be less than 255. Zero through 80 are reserved numbers. For reference, *1* implies a printer/plotter spooler queue, *2* implies an RJE queue, and *3* implies a batch queue. An example of a Queue.Index entry for plotters is:

```
PLOT/[SYS]<spl>PLOT.Queue/1/1
```

To create a Queue.Index file, define one queue entry for a generic printer/plotter, a Control Queue for each specific printer/plotter, and a Spoolerstatus Queue.

A Control Queue is the *user-defined name*, with *Control* appended to it, that will be used by the application to access a specific printer/plotter. A control name for each specific printer/plotter is needed. An example of a Control Queue is:

```
2PenPlotterCONTROL/[SYS]<spl>2PenPlotterCONTROL.Queue/1/1
```

A Spoolerstatus Queue is a queue by the name of Spoolerstatus.

An example of a new Queue.Index file is:

```
PLOT/[SYS]<spl>PLOT.Queue/1/1
2PenPlotterCONTROL/[SYS]<spl>2PenPlotterCONTROL.Queue/1/1
SPOOLERSTATUS/[SYS]<spl>SPOOLERSTATUS.Queue/1/1
```

SplCnfg.Sys File

An entry in the [Sys]<Sys>SplCnfg.sys file must be made if the peripheral is to be a spooled device. The format for this is:

Channel/Name/QueueName/ConfigurationFileSpec/Priority/Banner

Channel A single character code to designate the channel (port) to be used by the printer/plotter. The codes are as follows:

Code	Channel
0	Parallel
A	Serial A
B	Serial B
C	-

Name Must be the same QueueName prefix from the CONTROL name in the Queue.Index file (e.g., 2PenPlotter).

QueueName Must the same name as the generic printer/plotter name in the Queue.Index file (e.g., PLOT).

ConfigurationFileSpec The configuration file names (e.g., HPPlotterConfig.sys).

Priority Set to a value between 129 and 254.

Banner Determines if a banner page is printed at the beginning of each file or not. Default is *N* (no).

An example of an entry into the [Sys]<Sys>SplCnfg.sys is:

```
B/2PenPlotter/PLOT/[SYS]<sys>HPPlotterConfig.sys/130/N
```

Sample Programs

This section contains five sample programs. There is one Graphics Manager program for each language. Each program creates the same abstract line design. The BTOS calls, SetScreenVidAttr, and ResetFrame are included in the sample programs for screen manipulation. For more details about these two calls, refer to *BTOS Reference Volumes I and II*. Following these four programs, there is a Pascal program which uses Graphics Library calls to draw an illustration of a smiling cat.

To make your programs work in BASIC, COBOL or FORTRAN, you need to reconfigure the language so that the run file for the specific language includes graphics object modules. Complete the following steps where necessary. Refer to the appropriate language manual if you need more help. Also, these steps are explained in more detail in Section 9.

1 Make sure that the comment symbols are removed from the appropriate Table Entries (under the Graphics section) in the language.asm file. Only the calls used in the sample programs need to be uncommented. This applies to BASIC, COBOL and FORTRAN.

2 Assemble the appropriate "language".asm file. Type **Yes** to the prompt GRAPHICS. This applies to BASIC, COBOL and FORTRAN.

3 For BASIC and COBOL, recreate the run file for the language by submitting the file "StartCOBOLLink.sub" or "StartBASICLink.sub".

   For FORTRAN, when you link the sample program, be sure to include the file "<sys>ForGen.obj" in the object module field of the Link form.

4 Run the program.

COBOL Graphics Manager Program

```
*****************************************************************
COBOL Graphics Manager Program
Remember, you must have at least 640 K to run COBOL!
*****************************************************************
       WORKING-STORAGE SECTION.
       01  ERC                 PIC 9(4) COMP VALUE ZEROES.
       01  ERC-TEMP            PIC 9(4).
       01  xCenter             PIC 9(4) COMP.
       01  yCenter             PIC 9(4) COMP.
       01  WX2                 PIC 9(4) COMP VALUE 0.
       01  WY2                 PIC 9(4) COMP VALUE 0.
       01  Zero-Value          PIC 9(4) COMP VALUE 0.

*****  Below is the structure that is returned from GetRasterInfo.

       01  RInfo.
           05  cntPlanes       PIC 9(4) COMP.
           05  BytesPerLine    PIC 9(4) COMP.
           05  wWidth          PIC 9(4) COMP.
           05  wHeight         PIC 9(4) COMP.
           05  wVWidth         PIC 9(4) COMP.
           05  wVHeight        PIC 9(4) COMP.
           05  pPlane          PIC 9(9) COMP OCCURS 3 TIMES.

       PROCEDURE DIVISION.

*****  Initialize Graphics.

           CALL "&INITSCREENGRAPHICS" USING ERC.
           IF ERC IS NOT EQUAL TO 0
              GO TO Mistake.


*****  The number of pixels varies on the different workstations.
*      A call to GetRasterInfo is used to make this program run
*      correctly on all workstations, regardless of the differences
*      in the number of pixels.  In other words, this call allows
*      this device dependent program to run independently of the
*      workstation hardware.

           CALL "&GETRASTERINFO" USING ERC, RInfo.
           IF ERC IS NOT EQUAL TO 0
              GO TO Mistake.

*****  These ConvertWord procedures must be called in order
*      to use the information returned by GetRasterInfo as
*      integer values.

           PERFORM Convert-Para.

*****  Calculate the center X and Y coordinates of the screen
*      with the information returned by GetRasterInfo.
```

```
           DIVIDE wVWidth BY 2 GIVING xCenter.
           DIVIDE wVHeight BY 2 GIVING yCenter.

*****   Draw lines from the center of the screen outwards using
*       the information returned from GetRasterInfo as the end of
*       the X vector and then the end of the Y vector.

*****   To produce the Moire pattern, one of the coordinates in
*       DrawScreenLine must be a variable AND must be either increased
*       or decreased in "steps" greater than 1. The "stepping factor"
*       needed to produce the pattern varies depending on the type
*       of workstation.

           PERFORM Draw-Para VARYING WX2 FROM 0 BY 5
                                        UNTIL WX2 IS > wVWidth.
           PERFORM Draw-Para2 VARYING WY2 FROM 0 BY 5
                                         UNTIL WY2 IS > wVHeight.
           STOP RUN.
*****************************************************************
       Convert-Para.
          Call "&CONVERTWORD" USING wVWidth,  wVWidth.
          Call "&CONVERTWORD" USING wVHeight, wVHeight.
*****************************************************************
       Draw-Para.
          CALL "&DRAWSCREENLINE" USING ERC,
                                 xCenter,yCenter,WX2,Zero-Value.
          IF ERC IS NOT EQUAL TO 0
              GO TO Mistake.
          CALL "&DRAWSCREENLINE" USING ERC,
                                 xCenter,yCenter,WX2,wVHeight.
          IF ERC IS NOT EQUAL TO 0
              GO TO Mistake.
*****************************************************************
       Draw-Para2.
          CALL "&DRAWSCREENLINE" USING ERC,
                                 xCenter,yCenter,Zero-Value,WY2.
          IF ERC IS NOT EQUAL TO 0
              GO TO Mistake.
          CALL "&DRAWSCREENLINE" USING ERC,
                                 xCenter,yCenter,wVWidth,WY2.
          IF ERC IS NOT EQUAL TO 0
              GO TO Mistake.
*****************************************************************
       Mistake.
          MOVE ERC TO ERC-TEMP.
          DISPLAY ERC-TEMP UPON CRT.
          STOP RUN.

*****************************************************************
End of COBOL Graphics Manager Program
*****************************************************************
```

Pascal Graphics Manager Program

```
  (This is an example of a  program in Pascal.)

PROGRAM Demo;

TYPE
   POINTER = ADS OF WORD;
(  This is the structure that is returned from GetRasterInfo.)

   RasterInfoType = RECORD
     cntPlanes          : Word;
     wBytesPerLine      : Word;
     wWidth             : Word;
     wHeight            : Word;
     wVirtualWidth      : Word;
     wVirtualHeight     : Word;
     pPlane             : Array [0..2] of POINTER;
   END;

VAR
  RasterStuff           : RasterInfoType;
  xCenter               : Word;
  yCenter               : Word;
  J                     : Word;
  erc                   : Word;
  DONE                  : Boolean;

FUNCTION  InitScreenGraphics                          : WORD; EXTERN;
FUNCTION  GetRasterInfo (pbRasterInfo:POINTER)        : WORD; EXTERN;
FUNCTION  DrawScreenLine (wX1,wY1,wX2,wY2:WORD)       : WORD; EXTERN;
FUNCTION  SetScreenVidAttr(iAttr:INTEGER;fOn:Boolean): WORD; EXTERN;

PROCEDURE ERROREXIT(erc:Word);                                EXTERN;

PROCEDURE Error(erc:Word);
BEGIN
  If erc > 0 then ErrorExit(erc);
END;

BEGIN

  DONE := FALSE;

(   Initialize Graphics.                                      )

  Error(InitScreenGraphics);

(   This turns off the character video refresh, so that only  )
(   graphics images are displayed.                            )

  Error(SetScreenVidAttr(1,FALSE));

(   The number of pixels varies on the different workstations. )
(   A call to GetRasterInfo is used to make this program run   )
```

```
(   correctly on all workstations, regardless of the differences  )
(   in the number of pixels.  In other words, this call allows    )
(   this device dependent program to run independently of the     )
(   workstation hardware.                                         )
  Error(GetRasterInfo(ADS RasterStuff));

(   Calculate the center X and Y coordinates of the screen with   )
(   the information returned by GetRasterInfo.                    )

  xCenter := RasterStuff.wVirtualWidth DIV 2;
  yCenter := RasterStuff.wVirtualHeight DIV 2;

(   Draw lines from the center of the screen outwards             )
(   using the information returned from GetRasterInfo as          )
(   the end of the X vector and then the end of the Y vector.     )

(   To produce the Moire pattern, one of the coordinates in       )
(   DrawScreenLine must be a variable AND must be either          )
(   increased or decreased in "steps" greater than 1.  The        )
(   "stepping factor" needed to produce the pattern varies        )
(   depending on the type of workstation.                         )

  J :=0;

  REPEAT

    Error(DrawScreenLine(xCenter, yCenter, J, 0));
    Error(DrawScreenLine(xCenter, yCenter, J, RasterStuff.wVirtualHeight));

    J := J + 5;

    If J > RasterStuff.wVirtualWidth THEN

       DONE := TRUE;

  UNTIL DONE;

  J :=0;

  DONE :=FALSE;

  REPEAT

    Error(DrawScreenLine(xCenter, yCenter, 0,J));
    Error(DrawScreenLine(xCenter, yCenter, RasterStuff.wVirtualWidth, J));

    J := J + 5;

    If J > RasterStuff.wVirtualHeight THEN

       DONE := TRUE;

  UNTIL DONE;
END.
```

BASIC Graphics Manager Program

```
10   REM    This is an example of a Graphics Manager program in BASIC.
20       OPTION BASE 0
30   REM
40   REM    Set up an array to hold the structure that is
50   REM    returned from the call GetRasterInfo.
60   REM
70       DIM RASTERINFO%[12]
80   REM
90   REM    Initialize Graphics.
100  REM
110      ERC% = INITSCREENGRAPHICS()
120      IF ERC% <> 0 THEN ERROREXIT(ERC%)
130  REM
140  REM    This turns off the character video refresh, so that
150  REM    only graphics images are displayed.
160  REM
170      ERC% = SETSCREENVIDATTR(1,0)
180      IF ERC% <> 0 THEN ERROREXIT(ERC%)
190  REM
200  REM    The number of pixels varies on the different workstations.
210  REM    A call to GetRasterInfo is used to make this program run
220  REM    correctly on all workstations, regardless of the differences
230  REM    in the number of pixels.  In other words, this call allows
240  REM    this device dependent program to run independently of the
250  REM    workstation hardware.
260  REM
270      ERC% = GETRASTERINFO(PTR(RASTERINFO%[0]))
280      IF ERC% <> 0 THEN ERROREXIT(ERC%)
290  REM
300  REM    Calculate the center X coordinate of the screen
310  REM    with the information returned by GetRasterInfo.
320  REM
330      XCENTER% = RASTERINFO%[4] / 2
340  REM
350  REM    Calculate the center Y coordinate of the screen
360  REM    with the information returned by GetRasterInfo.
370  REM
380      YCENTER% = RASTERINFO%[5] / 2
390  REM
400  REM    Draw lines from the center of the screen
410  REM    outwards using the information returned from
420  REM    GetRasterInfo as the end of the X vector and then
430  REM    the end of the Y vector.
435  REM
440  REM    To produce the Moire pattern, one of the coordinates in
450  REM    DrawScreenLine  must be a variable AND must be either
460  REM    increased or decreased in "steps" greater than 1.  The
470  REM    "stepping factor" needed to produce the pattern varies
480  REM    depending on the type of workstation.
```

```
490  REM
500       FOR J% = 0 TO RASTERINFO%[4] STEP 5
510       ERC% = DRAWSCREENLINE(XCENTER%,YCENTER%,J%,0)
520       IF ERC% <> 0 THEN ERROREXIT(ERC%)
530       ERC% = DRAWSCREENLINE(XCENTER%,YCENTER%,J%,RASTERINFO%[5])
540       IF ERC% <> 0 THEN ERROREXIT(ERC%)
550       NEXT J%
560       FOR J% = 0 TO RASTERINFO%[5] STEP 5
570       ERC% = DRAWSCREENLINE(XCENTER%,YCENTER%,0,J%)
580       IF ERC% <> 0 THEN ERROREXIT(ERC%)
590       ERC% = DRAWSCREENLINE(XCENTER%,YCENTER%,RASTERINFO%[4],J%)
600       IF ERC% <> 0 THEN ERROREXIT(ERC%)
610       NEXT J%
999       END
```

FORTRAN Graphics Manager Program

```
C This is an example of a Graphics Manager program in FORTRAN

      Subroutine ChkErc(ierc)
         If (ierc.NE.0) RETURN
         Call tmerex(ierc)
      End

      PROGRAM ID DEMO

      Implicit integer (g,x,t,y,v)
      Implicit integer*2 (r)
      external grisgr
      external grdsin
      external grrain
      external tmerex
      external vdstsa

C Set up an array to hold the structure that is
C returned from the call GetRasterInfo.

      dimension RInfo(12)

C Initialize Graphics.

      Call ChkErc(grisgr())

C This turns off the character video refresh, so only graphics
C images are displayed.

      Call ChkErc(vdstsa(1,0))

C The number of pixels varies on the different workstations.
C A call to GetRasterInfo is used to make this program run
C correctly on all workstations, regardless of the differences
C in the number of pixels.  In other words, this call allows
C this device dependent program to run independently of the
C workstation hardware.

      Call ChkErc(grrain(RInfo(1)))

C Calculate the center X and Y coordinates of the screen with
C the information returned by GetRasterInfo.

      xCenter = RInfo(5)/2
      yCenter = RInfo(6)/2

C Draw lines from the center of the screen outwards, using
C the information returned from GetRasterInfo as the end of the
C X vector and then the end of the Y vector.
```

```
C To produce the Moire pattern, one of the coordinates in
C DrawScreenLine must be a variable AND must be either
C increased or decreased in "steps" greater than 1.  The
C "stepping factor" needed to produce the pattern varies
C depending on the type of workstation.

      do 10 j=0,Rinfo(5),5

      Call ChkErc(grdsin(xCenter,yCenter,j,0))

      Call ChkErc(grdsin(xCenter,yCenter,j,Rinfo(6)))

10    continue

      do 20 j=0,Rinfo(6),5

      Call ChkErc(grdsin(xCenter,yCenter,0,j))

      Call ChkErc(grdsin(xCenter,yCenter,Rinfo(5),j))

20    continue

      stop
      end
```

Pascal Graphics Library Program

```
  ( This is an example of a Graphics Library program in Pascal)
   ( Remember to include the Graphics Library in the Link form. )

PROGRAM Demo(INPUT,OUTPUT);

TYPE
   POINTER = ADS OF WORD;

VAR
  GraphicsMem           : POINTER;
  Erc                   : WORD;
  MSG                   : LString(80);

CONST
  PicMode=#6D77;

FUNCTION  AddObject (pbObjectName:POINTER;cbObjectName:WORD;
                     rXMin,rYMin,rXMax,rYMMax:REAL)          :WORD; EXTERN;
FUNCTION  AllocMemorySL(cBytes:WORD;ppSegmentRet:POINTER)    :WORD; EXTERN;
FUNCTION  CloseObject                                        :WORD; EXTERN;
FUNCTION  ClosePicture(fSave:BOOLEAN)                        :WORD; EXTERN;
FUNCTION  DrawArc(rXCenter,rYCenter,rSRadius,rAng1,rAng2:REAL):WORD; EXTERN;
FUNCTION  DrawCircle(rXCenter,rYCenter,rSRadius:REAL)        :WORD; EXTERN;
FUNCTION  DrawLine(rX1,rY1,rX2,rY2:REAL)                     :WORD; EXTERN;
FUNCTION  FillRectangle(rXMin,rYMin,rXMax,rYMax:REAL;
                        bFillType:BYTE)                      :WORD; EXTERN;
FUNCTION  InitGraphics                                       :WORD; EXTERN;
FUNCTION  Move(rX,rY:REAL)                                   :WORD; EXTERN;
FUNCTION  OpenPicture(pbPicName:POINTER;cbPicName:WORD;
                      pbPassWord:POINTER;cbPassWord:WORD;Mode:WORD;
                      pMemory:POINTER;cParasMemory:WORD)     :WORD; EXTERN;
FUNCTION  ResetFrame(lFrame:WORD)                            :WORD; EXTERN;
FUNCTION  SetCharacterSize(rSChars:REAL)                     :WORD; EXTERN;
FUNCTION  SetDrawingMode(lMode:WORD)                         :WORD; EXTERN;
FUNCTION  SetLabelOrigin(bLorg:BYTE)                         :WORD; EXTERN;
FUNCTION  WriteTextString(pbString:POINTER;cbString:WORD)    :WORD; EXTERN;

PROCEDURE ERROREXIT(erc:Word)                                      ; EXTERN;

PROCEDURE Error(erc:Word);
BEGIN
  If erc > 0 then ErrorExit(erc);
END;

BEGIN

   ( Allocate short lived memory for the picture file workarea. )

  Error(AllocMemorySL(#F000, ADS GraphicsMem));

       ( Initialize graphics. )
```

```
Error(InitGraphics);

     ( Open a picture in write mode. )

Error(OpenPicture(ADS 'DemoPic',7,ADS ' ',0,PicMode,GraphicsMem,#0F00));

     ( Add an object and set user coordinates. )

Error(AddObject(ADS 'ObjectOne',9,0,0,100,100));

( Clear the screen so that only the graphics images will appear.)

Error(ResetFrame(0));
Error(ResetFrame(1));
Error(ResetFrame(2));

     ( Begin drawing the image.  This program uses the following drawing  )
     ( functions; DrawCircle, FillRectangle, DrawArc, Move, and DrawLine.)
     ( SetCharacterSize, SetLabelOrigin and Move are used to set up the )
     ( size and position of a message that will appear on the screen. )

Error(DrawCircle(50,50,40));

Error(DrawCircle(40,70,10));

Error(DrawCircle(60,70,10));

Error(DrawCircle(46,70,2));

Error(DrawCircle(54,70,2));

Error(FillRectangle(45,40,55,50,5));

Error(DrawArc(50,35,20,3.14,6.28));

Error(SetCharacterSize(0.5));

Error(SetLabelOrigin(2));

Error(Move(30,05));

MSG:='Press <RETURN> to exit program';

Error(WriteTextString(ADS MSG[1], WRD(MSG[0])));

Error(DrawLine(45,42,20,36));

Error(DrawLine(45,45,20,42));

Error(DrawLine(45,50,20,56));
```

```
  Error(DrawLine(55,42,80,36));

  Error(DrawLine(55,45,80,42));

  Error(DrawLine(55,50,80,56));

       ( Wait for input from the keyboard.                    )

  ReadIn;

       ( Close the object and close the picture.              )

  Error(CloseObject);

  Error(ClosePicture(FALSE));

END.
```

Drawing Modes and Fill Patterns

Drawing Modes

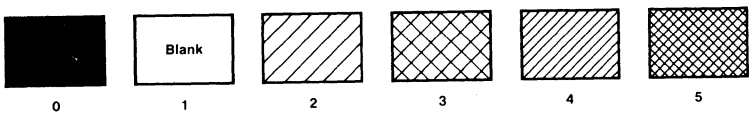
The table below demonstrates the effect of the four drawing modes on a three plane color system. The initial values for the planes, the line type and modes are indicated. Drawing is performed in color three. The background color on Burroughs graphics workstations is 0. Drawing modes cannot be printed or plotted; they can only be displayed. See Figure B-2 for examples of the drawing modes.

Plane	Value	Set	Clear	XOR	Replace
0	01010101	01111101	00010100	00111100	01101001
1	00110011	01111011	00010010	01011010	01101001
2	00001111	00000110	00000110	01101110	00000000

Line Type: 01101001
iColor: 3
Background color:

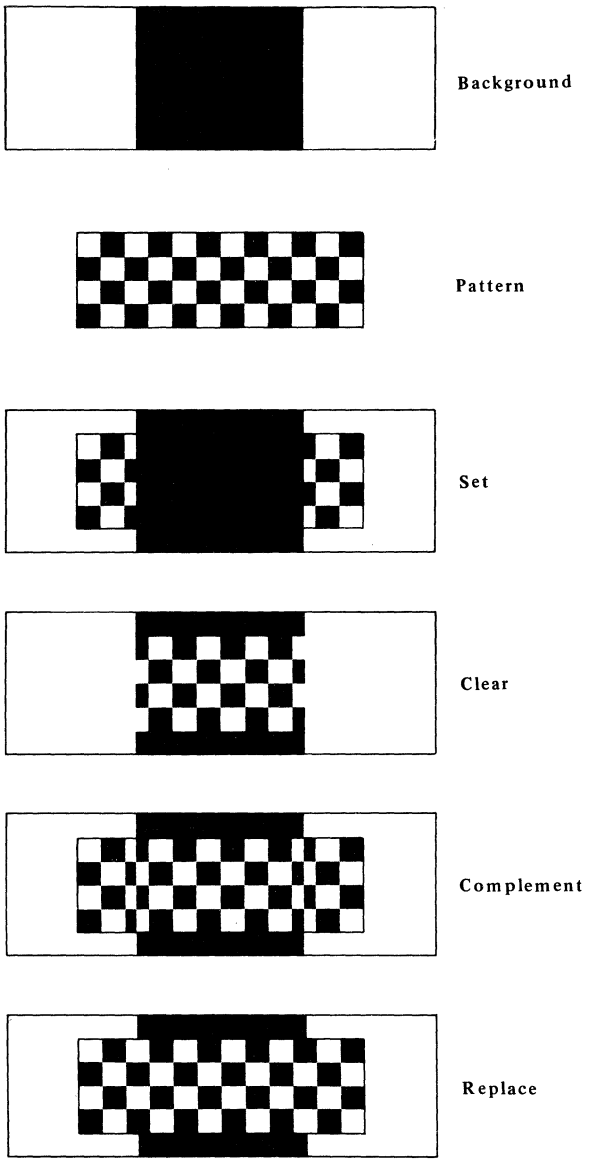
Fill Patterns

Figure B-1 **Fill Patterns**



The current drawing mode and line type do not affect the appearance of the vectors drawn in a fill pattern. For example, a rectangle that is drawn in a solid fill pattern remains solid, regardless of which drawing mode or line type is current. However, any vectors drawn over that rectangle will be drawn in the current drawing mode and line type.

Figure B-2 **Drawing Modes**



Graphics Manager Virtual and Physical Pixel Resolution

Table C-1 **80-Column Mode Pixel Resolution**

Workstation	Screen Size	Physical	Virtual
B 21		(431),(318)	(431), (318)
B 22		(655),(509)	(655), (509)
B 25		(719),(347)	(1439),(1043)
B 27	12-Inch Mono	(719),(479)	(1328), (959)
B 27	15-Inch Mono	(719),(479)	(1247), (959)
B 27	14-Inch Color	(719),(479)	(1247), (959)
B 28		(719),(347)	(1439),(1043)

Table C-2 **132-Column Mode Pixel Resolution**

Workstation	Screen Size	Physical	Virtual
B 22		(655),(509)	(655), (509)
B 27	12-Inch Mono	(791),(479)	(1339), (959)
B 27	15-Inch Mono	(791),(479)	(1266), (959)

Note: *Because the first coordinate in a display is (0,0), not (1,1), the actual physical and virtual resolution of each machine is one pixel higher than noted in the previous tables. For example, the actual physical resolution for the B 25 in 80-column mode is 720 by 348. The following illustration indicates a screen display where (x,y) is a physical or virtual coordinate from one of the previous tables.*

Note: *Because all machines have different virtual pixel resolutions, make your programs as device-independent as possible; use GetRasterInfo (a Graphics Manager request, see Section 7) to determine virtual and physical pixel resolutions.*

Graphics Library Aspect Ratios World and NDC Coordinates

Table D-1 **80-Column Mode Coordinates**

Workstation	Screen Size	World Coordinate	Normalized Device Coordinate
B 21		(100.0),(73.78)	(1.000),(0.7378)
B 22		(100.0),(77.70)	(1.000),(0.7770)
B 25		(100.0),(72.50)	(1.000),(0.7250)
B 27	12-Inch Mono	(100.0),(72.23)	(1.000),(0.7223)
B 27	15-Inch Mono	(100.0),(76.92)	(1.000),(0.7692)
B 27	14-Inch Color	(100.0),(76.92)	(1.000),(0.7692)
B 28		(100.0),(72.50)	(1.000),(0.7250)

Table D-2 **132-Column Mode Coordinates**

Workstation	Screen Size	World Coordinate	Normalized Device Coordinate
B 22		(100.0),(77.70)	(1.000),(0.7770)
B 27	12-Inch Mono	(100.0),(71.64)	(1.000),(0.7164)
B 27	15-Inch Mono	(100.0),(75.76)	(1.000),(0.7576)

Note: *The coordinates listed in the previous tables are based on the first coordinate being (0,0). The following illustration indicates a screen display where (x,y) is a coordinate from one of the previous tables.*

System Walkthrough: Graphics Library

Graphics Library (Graphics.lib) contains device-independent routines that must be linked into the graphics application program. The Graphics Manager contains the device-dependent requests. Graphics Library routines use these requests for all screen drawing commands.

Your first call must always be to InitGraphics. InitGraphics initializes variables and sets up the hardware for future use.

To use your application with a B 27 workstation, the next call must be SetColumnMode. This procedure sets up the current column mode. If you do not call SetColumnMode, the application assumes the column mode used in the most previous task; there is no default mode. If you never use a B 27 workstation, you do not need to make this call.

After issuing InitGraphics and SetColumnMode calls, your application needs to issue the picture procedures, listed below. These procedures perform such specific tasks as open, write, save, or display the current picture. Your application can also get the number of objects contained within the current picture. After opening a picture, you can then manipulate the objects within that picture.

Attribute procedures set the specific line type, color, drawing mode or current palette.

Drawing procedures draw lines and arcs.

Transformation procedures translate the current object's size, shape or position on the display screen.

Label procedures define the position, attributes, and transformations of a label. To completely define a label, you must also use the Text and Font procedures.

Text procedures create text strings associated with a label.

Font procedures specify which font is to be used for the text string.

Viewing procedures set both viewports and windows. These procedures are used to operate on pictures as a whole, not on the individual objects within the picture.

Cursor procedures position the cursor on the display screen and can be set to refer to an object, a picture, or the viewport.

Initialization procedures establish the type of output device, such as a printer or plotter, and the type of material to be used, such as transparencies or paper.

User-Replaceable procedures are currently used to interrupt printer procedures, and to let the user know to load paper and change pens. They can be replaced by the user to include other actions by replacing these routines in the Graphics.Lib library with the user's own procedures. Applications need to link in their own routines in place of one of the existing routines, and to use the Librarian command to replace the modules existing in Graphics.Lib itself.

Replaceable modes are:

GrfLdp: Replace this module with your own for messages instructing you to load paper or transparencies.

GrfSpn: Replace this module with your own for halting the plotter while changing one of its pens.

GrfRik: Replace this module with your own for an interupt to the DisplayPicture procedure.

An explicit procedure for ending the graphics session does not exist. The graphics session ends when the application is complete.

Label Structure

The label structure is used to modify an existing label. The current label is moved into this workarea structure and the modifications are made in the workarea. After all modifications are completed, the workarea is then copied 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 F-1 shows the label structure attributes

Table F-1 **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	real	the label origin of the label.
bPen	byte	the pen number for the label for plotter output, or in 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	byte	the actual text of the label, which(cbLabel) is chLabel bytes long.

Status Codes

Decimal Value	Meaning
7600	During Graphics Manager initialization, indicates that graphics is not available on this workstation. Applications requesting a graphics page (through a request to either InitScreenGraphics or InitAdditionalGraphicsScreen) receive this error when no graphics pages are left to allocate.
7601	A graphics operation was invoked without first initializing graphics.
7602	An error occured internal to the Graphics Library.
7603	Raster Operations are not available.
7604	Invalid polygon fill algorithm.
7605	Point, line, curve or rectangle coordinates outside of the user defined viewport. This error is only returned if the entire object is outside of the defined viewport (entire object clipped).
7606	The Graphics Manager has already been installed as a system service.
7607	There are currently no graphics screens available.
7608	Invalid Task User. Attempted to run graphics application in Batch Mode.
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.
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 write (vectors and labels) the open object when the picture was opened in read or modify mode.

Decimal Value	Meaning
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.
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.
7648	Parameters were specified which are not contained within the given viewport boundaries.
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 device was specified.
7800	Invalid command screen number passed to SetCommandScreen.
7801	Invalid visible screen number passed to SetVisibleScreen.
7802	Requested function is not available on your workstation's hardware.
7810	Invalid parameters passed to ClearScreenRectangle.
7811	Invalid parameters passed to DrawScreenArc.
7812	Invalid parameters passed to DrawScreenLine.
7813	Invalid parameters passed to FillScreenRectangle.
7814	Invalid soft pattern index passed to LoadSoftPattern.
7815	Invalid pixel address.
7816	Invalid screen color passed to SetScreenColor.
7817	Invalid Drawing Mode requested.
7818	Invalid Line Type requested.
7830	Invalid Color Mapper requested.
7831	Invalid Color Mapper index.

Decimal Value	Meaning
7840	Attempt to change graphics width to an invalid width.
7845	A buffer size in a DrawScreenBuffer command cannot be correct, given the commands in the buffer.
7846	DrawScreenBuffer encountered an invalid request code in a command buffer.

Glossary

Aspect Ratio The aspect ratio is the ratio of height to width of a pixel.

Background Color The color to which the screen is set during initialization and clearscreen functions. In a polychromatic system, the zeroeth entry in the palette. In a monochromatic system, the background is defined to be black (pixels of display value zero). See Color Palette.

BTOS Burroughs Operating System.

BTOS Systems Generic term used to refer to all Burroughs workstations or master systems running with the BTOS operating system.

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

Clipping Clipping describes the process of finding if a line (or polygon) intersects a polygon. In Graphics for BTOS Systems, clipping takes place if a line, curve, or polygon is drawn outside the defined viewport. The part of the line, curve, or polygon which falls outside of the viewport is not drawn.

Color Palette The color palette is a set of colors (or gray levels) that can be used for color (intensity) specification on a color (gray level) graphics workstation display or on a plotter or color printer output. Background and Foreground are indices to entries in the color palette.

Command Screen (Command Page) Graphics memory page where graphics operations are performed. This can be different than the visible screen. Each graphics task must have a command screen (specified by SetCommandScreen) which is one of the display screens owned by the task. If a task has only one graphics screen, the command and visible screen must be the same.

Device-Dependent procedures In Graphics for BTOS Systems, this term refers to routines that directly access the Burroughs graphics hardware. See Graphics Manager.

Device-Independent procedures Procedures that can run interchangeably between workstations in a system. See Graphics Library.

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 the current command screen. Modes which are supported:

- Set (OR)
- Clear (Reset [set with background color])
- Complement (XOR)
- Replace

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. Both vector and raster fonts can be defined. The names of all the vector font files available for use must be in the file "Graphics.Fonts," which is located on the master hard disk or your local hard disk, depending on where it was installed.

Foreground Color The foreground is the color drawn for a solid line in set mode, and is referred to as the current color. On a color workstation, the foreground color is treated as an index into the color palette. See color palette.

Graphics Control Block (GCB) This is an area of the Video Control Block (VCB) that contains all information about a task's graphics display, including the current viewport, palette, line type, drawing mode, foreground colors, current display and command pages, the total number of initialized graphics pages, etc.

Graphics Library (Graphics.Lib) The entire set of routines, data structures, and support code which are referred to as the device-independent library. These routines use Graphics Manager to perform their graphics memory (page) allocation, drawing, and bitmap related requests. Graphics.Lib is linked into application programs. These device-independent procedure are designed to support object definition, picture storage and retrieval, translating and scaling, and output to various hardware devices, routines which will work on all BTOS graphics hardware. The device independent procedures support a user defined coordinate system for drawing. They work by transforming the user's coordinate space into the physical coordinate space provided by the user's graphics hardware, and requesting drawing and other graphics operations from the Graphics Manager. These procedures can be linked into any application which requires them.

Graphics Manager This is the collection of routines that handle graphics hardware in the system. The Graphics Manager responds to requests from the operating system to inform it of task termination, foreground task change, and changes in a task's 80- or 132- column display format, as well as service graphics drawing requests from client tasks. The user accessible commands use hardware based coordinates, operate with word parameters, and provide no transformation, windowing, scaling or storage capabilities. These procedures execute at higher speeds than the device independent procedures due to these restrictions, but programs using them typically execute on only a single type of graphics hardware.

Graphics Page The memory that contains the bitmap for the graphics screen is called the graphics page. In a monochrome system, for every physical pixel on the screen there is one bit that controls that pixel. Thus, the graphics page contains one plane. In a color system, for every physical pixel on the screen, there are three bits controlling that pixel (allowing eight colors to be displayed simultaneously on the screen). Thus, the graphics page contains three planes. Some systems can have more than one graphics page, but only one graphics page controls the physical screen at a time.

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 line is drawn.

Normalized Device Coordinate System The normalized device coordinate system is a Graphics Library coordinate system used to reference positions on the video display screen 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 displayed at a later time.

Pixels Stands for picture elements. The smallest dot which can be drawn on the screen.

Process A process is the basic entity which competes for access to the processor(s) and which the BTOS operating system schedules for execution. Associated with a process is the address (CS:IP) of the next instruction to execute on behalf of the processor, a copy of the data to be loaded into the processor registers before control is returned to this process, a default response exchange, and a stack.

Request Block A data structure which contains the specification and the parameters of a desired system service.

System Service An operating system process that receives and responds to requests from client processes. System services make use of request blocks to transfer data between procedures, handle resource allocation and provide services such as file system, video, graphics and queue management. They may either be an integral part of the operating system (like the file system) or they may be installed after system initialization (like the Graphics Manager).

Task A task consists of executable code, data, and one or more processes. The code and data can be unique to the task or shared with other tasks. A task can coexist with or replace other application tasks.

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

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 Graphics Library 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-Replaceable Procedures User-Replaceable procedures are Graphics Library routines called by the graphics software that can be replaced by user-implemented versions.

Vector A line drawn between 2 given points.

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

Video Control Block (VCB) Contains information about a task's character video and graphics display.

Viewport The viewport is the portion of the video display screen that defines where any object may be displayed.

Virtual Coordinate System A system that simulates square pixels for the Graphics Manager. The procedures that use this system internally translate the virtual coordinates into real coordinates. When virtual coordinates are used, horizontal and vertical lines of the same length will appear to be the same length on the screen. If virtual coordinates were not used, a horizontal line 20 pixels long would not be the same size as a vertical line 20 pixels long.

Visible Screen Specifies which display memory page is to be seen. A tasks visible screen may be independent of its command screen. Each graphics task must own a visible screen (specified by GCB.rgbScreens(GCB.bVisibleScreen)). If a task has only one graphics screen, the command and visible screen must be the same.

Window The window is the portion of the world coordinate system that defines what is to be displayed in the current viewport. Coordinate positions outside the window are clipped.

World Coordinate System The world coordinate system is a Graphics Library coordinate system used internally by the device-independent Graphics Library software to map objects to the current command screen.

Title: _____

Form Number: _____ Date: _____

Burroughs Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information.

Please check type of suggestion: ☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments: _____

_____

_____

_____

Name _____

Title _____

Company _____

Address _____
Street City State Zip

Telephone Number ( ) _____
Area Code

Title: _____

Form Number: _____ Date: _____

Burroughs Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information.

Please check type of suggestion: ☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments: _____

_____

_____

_____

Name _____

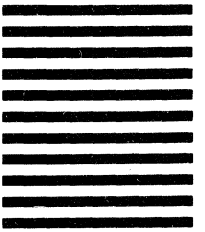
Title _____

Company _____

Address _____
Street City State Zip

Telephone Number ( ) _____
Area Code

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY CARD

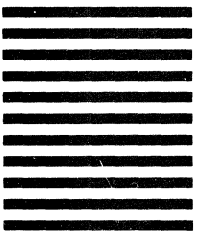
FIRST CLASS PERMIT NO. 817 DETROIT, MI 48232

POSTAGE WILL BE PAID BY ADDRESSEE

Burroughs Corporation
Production Services – East
209 W. Lancaster Avenue
Paoli, Pa 19301 USA

ATTN: Corporate Product Information

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 817 DETROIT, MI 48232

POSTAGE WILL BE PAID BY ADDRESSEE

Burroughs Corporation
Production Services – East
209 W. Lancaster Avenue
Paoli, Pa 19301 USA

ATTN: Corporate Product Information