Programmer's Guide to DOMAIN Graphics Primitives

Order No. 003245
Revision 01
Software Release 8.0

apollo computer inc.

Programmer's Guide to DOMAIN Graphics Primitives

Order No. 003245
Revision 01
Software Release 8.0

apollo computer inc.

Printed in U.S.A.

First printing: April, 1984

This document was formatted using the FMT tool
distributed with the Apollo DOMAIN Computer system.

PREFACE

The Programmer's Guide to DOMAIN Graphics Primitives describes the DOMAIN* graphics primitives system and its user-callable routines.

Audience

This manual is written for programmers who use the DOMAIN graphics primitives to develop application programs. It is assumed that users of this manual have some knowledge of computer graphics and have experience in using the DOMAIN system.

Organization of this Manual

This manual contains twelve chapters.

Chapter 1 introduces the graphics primitives system and its use on a DOMAIN node.

Chapter 2 describes the display configurations, formats, and modes within which the graphics routines can operate.

Chapter 3 describes bitmap structure and pixels in relation to memory representation. The chapter includes discussion of bitmaps in display and main memory, bitmap routines, and bitmap attributes.

Chapter 4 describes the components and uses of a color map. The distinctive characteristics of a color map in imaging format are presented.

Chapter 5 explains the uses of current position and describes drawing, filling, and text operations.

Chapter 6 describes three types of bit block transfers (BLTs) and their uses. The differences between calls to the graphics primitives BLTs and display driver BLTs are also presented. Examples of BLT operations are included.

Chapter 7 describes the uses of cursor control and of input operations. Input operations include routines that enable graphics programs to accept input from various input devices such as a keyboard, button on a mouse or puck, touchpad, and cursor in a window transition.

* Distributed Operating Multi-Access Interactive Network

Chapter 8 describes the routines that graphics programs can call to perform direct input and output to windows. A sample program is included to show the input and output calls with other GPR routines.

Chapter 9 lists insert files, data structures with their argument type, and error messages.

Chapter 10 presents sample programs and techniques for using graphics primitives.

Chapter 11 describes graphics primitives routines in terms of format and parameters. The routines are organized alphabetically. A listing by functional category introduces the routines.

Chapter 12 describes graphics map files (GMF), including insert files, data structures, error messages, and user-callable routines.

Appendix A illustrates the 880 and low-profile keyboard and keyboard charts

Appendices B, C, and D present the insert files for Pascal, C, and FORTRAN

Additional Reading

For information on using the DOMAIN system, see the DOMAIN System Command Reference Manual. For information on the software components of the operating system and user-callable system routines, see the DOMAIN System Programmer's Reference Manual. For language-specific information, see the DOMAIN FORTRAN User's Guide and the DOMAIN Pascal User's Guide. For information on the high-level language debugger, see the Language Level Debugger Manual.

Documentation Conventions

Unless otherwise noted in the text, this manual uses the following symbolic conventions.

UPPERCASE	Uppercase words or characters in formats and command descriptions represent commands or keywords that you must use literally.
lowercase	Lowercase words or characters in formats and command descriptions represent values that you must supply.
[ ]	Square brackets enclose optional items in formats and command descriptions. In sample Pascal statements, square brackets assume their Pascal meanings.

{ } Braces enclose a list from which you must choose an item in formats and command descriptions. In sample Pascal statements, braces assume their Pascal meanings.

CTRL/Z The notation CTRL/ followed by the name of a key indicates a control character sequence. You should hold down the <CTRL> key while typing the character.

Change Bars in This Revision

This manual is a revision of 003245 Revision 00. This revision for SR8 includes new routines and technical changes. These additions and changes are indicated by a vertical bar in the margin of a page. Appendices B, C, and D present insert files for Pascal, C, and FORTRAN respectively; these appendices do not show any change bars.

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. In order to make it easy for you to communicate with us, we provide the User Change Request (UCR) system for software-related comments, and the Reader's Response form for documentation comments. By using these formal channels you make it easy for us to respond to your comments.

You can get more information about how to submit a UCR by consulting the DOMAIN System Command Reference Manual. Refer to the CRUCR (Create User Change Request) command. You can also get more information by typing:

```
$HELP CRUCR <RETURN>
```

For documentation comments, a Reader's Response form is located at the back of each manual.

SUMMARY OF TECHNICAL CHANGES

The Programmer's Guide to DOMAIN Graphics Primitives, Revision 01, includes new routines as well as changes and additions to existing routines.

The following new routines are included in this manual:

- External storage of bitmaps

  New routine opens a file of external storage of a bitmap

  GPR_$OPEN_BITMAP_FILE

- New attribute operations

  New routines for fill patterns, background of tile fill patterns, and line patterns:

  GPR_$SET_FILL_BACKGROUND_VALUE
  GPR_$INQ_FILL_BACKGROUND_VALUE

  GPR_$SET_FILL_PATTERN
  GPR_$INQ_FILL_PATTERN

  GPR_$SET_LINE_PATTERN
  GPR_$INQ_LINE_PATTERN

- New text operations

  New routines for setting the direction for writing text, creating a modifiable copy of a font, for setting character width, horizantal spacing, and space size:

  GPR_$SET_TEXT_PATH
  GPR_$INQ_TEXT_PATH

  GPR_$REPLICATE_FONT

  GPR_$SET_CHARACTER_WIDTH
  GPR_$INQ_CHARACTER_WIDTH

  GPR_$SET_HORIZONTAL_SPACING
  GPR_$INQ_HORIZONTAL_SPACING

  GPR_$SET_SPACE_SIZE
  GPR_$INQ_SPACE_SIZE

  GPR_$SET_TEXT_PATH
  GPR_$INQ_TEXT_PATH

GPR_$REPLICATE_FONT

- Line drawing operations

  New routines for arcs, circles, boxes, and splines:

  GPR_$ARC_3P

  GPR_$CIRCLE

  GPR_$DRAW_BOX

  GPR_$SPLINE_CUBIC_P

  GPR_$SPLINE_CUBIC_X

  GPR_$SPLINE_CUBIC_Y

- Filling operations

  New routine for drawing and filling circles:

  GPR_$CIRCLE _FILLED

Throughout the manual, change bars in the margin of the page indicate changes and additions.

CONTENTS

Illustrations

Tables

CHAPTER 1

INTRODUCTION TO GRAPHICS PRIMITIVES

This chapter briefly describes the uses and characteristics of the graphics primitives routines (GPR). The graphics primitives library is built into your DOMAIN system. The routines (primitives) that make up the library let you manipulate the least divisible graphic elements to develop high-speed graphics operations. These elements include lines and polylines, text fonts and values, pixel values, and display types.

The DOMAIN system also has an optional Core graphics package. The Core graphics package provides a high-level graphics environment in which to build portable graphics application systems. For a detailed description of Core graphics, see the Programmer's Guide to Apollo Core Graphics.

1.1 USES OF GRAPHICS PRIMITIVES

The graphics primitives include the following capabilities:

- Drawing lines, circles, and rectangles
- Acquiring text fonts and manipulating text
- Manipulating graphics with bit block transfers
- Filling polygon areas
- Accommodating input operations
- Setting attributes
- Enabling direct graphics operations
- Imaging with an extended color range
- Storing bitmaps externally

The GPR package uses the following components of the DOMAIN system.

- A display, 1024x800, 800x1024, or 1024x1024 visible pixels
- Display memory, 1024x1024 pixels, or 1024x2048 pixels
- Any portion of program memory
- A set of graphics primitive routines (all begin with "GPR")
- Optionally, the Display Manager

1.2 CHARACTERISTICS OF GRAPHICS PRIMITIVES

Graphics primitives are device dependent with respect to the display. However, they are independent of the various display environments. The operating system provides two other sets of calls to manipulate the display:

> Display Manager interface -- These program calls (which begin with PAD) allow you to manipulate pads and frames to display text. You cannot manipulate graphics using these calls.
>
> Display driver interface -- The Display Manager normally controls the DOMAIN display. All program requests to read or write the display are directed to the Display Manager, which in turn calls the display driver to perform screen operations for monochromatic displays. Most of the display driver (SMD) calls duplicate functions now provided by the graphic primitives package.

For a description of the calls to the Display Manager interface and the display driver interface see the DOMAIN System Programmer's Reference Manual.

GPR routines are independent of the display environments in two ways. First, you can run a program which includes GPR routines on any of the displays without modifying the program.

Second, graphics primitives routines can issue calls to either the Display Manager or the display driver. Therefore, if you use the graphics primitives routines, you can easily change program execution from one display mode to another by changing one option in the initialization routine GPR_$INIT.

CHAPTER 2

DISPLAY ENVIRONMENTS

This chapter describes the display configurations, formats, and modes within which the graphics routines can operate.

2.1 DISPLAY CONFIGURATIONS

All DOMAIN displays are bit-mapped raster-scan devices. Each node has one of three types of DOMAIN displays:

- color
- monochromatic portrait (vertical)
- monochromatic landscape (horizontal)

2.1.1 Monochromatic Displays

Monochromatic displays are either black and white or black and green. The monochromatic display memory is 1024 pixels wide and 1024 pixels high. There are two different monochromatic display devices: portrait and landscape. On the monochromatic portrait display, the left-most 800x1024 pixels are visible. On the monochromatic landscape display, the top 1024x800 pixels are visible. Figure 2-1 shows the monochromatic display configurations.

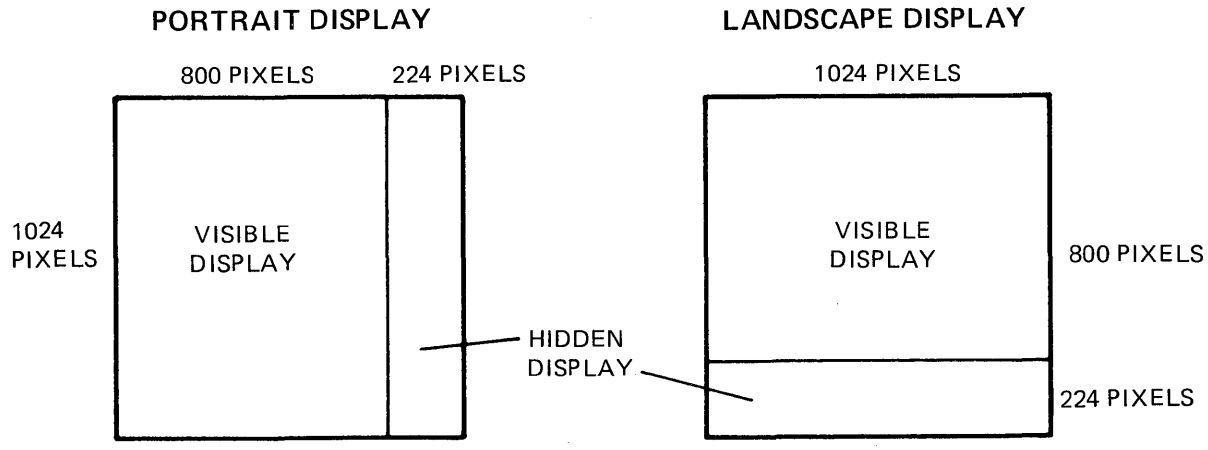


Figure 2-1. Display Memory Configurations: Monochromatic Displays

2.1.2 Color Displays

The color display has two types of format: interactive and imaging. The interactive formats support all GPR operations. The imaging formats support limited GPR operations, but provide the ability to display more colors. In imaging format, images are displayed with more bits per pixel than they are in interactive formats. This is useful for color correction in true-color imaging.

2.1.3 Hardware Configurations for Color Displays

Color displays have either a two- or three-board hardware configuration. For either configuration, programs can change the display between the two types of formats. Both types of format are available with both hardware configurations, as shown in Tables 2-1 and 2-2. This means that either configuration provides both interactive processing and more extensive color display.

Table 2-1. Two-Board Configuration for Color Display

Format	**Pixel Dimensions**		
	Visible Display	Hidden Display	Number of Colors
4-bit interactive (Default)	1024 x 1024	1024 x 1024	16
8-bit imaging	1024 x 1024	none	256

Table 2-2. Three-Board Configuration for Color Display

Format	Pixel Dimensions		
	Visible Display	Hidden Display	Number of Colors
8-bit interactive (Default) 24-bit imaging	1024 x 1024 512 x 512	1024 x 1024 512 x 512	256 16 million

Two-Board Configuration

The interactive 4-bit pixel format is the default for a two-board configuration. This means that 4 bits are used to assign a pixel value (color map index) to each pixel. This format allows sixteen different colors to appear on the screen at one time. The pixels are arranged 1024 x 1024 in visible display memory and 1024 x 1024 in hidden display. Interactive formats support all GPR operations.

Optionally, software can change a two-board configuration to an 8-bit imaging format, with 8 bits used to assign a pixel value (color map index) to each pixel. This format allows 256 colors to appear on the screen at one time, but requires only two boards. The pixels are arranged 1024 x 1024 in the visible display. There is no hidden display. Imaging formats support only limited GPR operations.

Three-Board Configuration

The interactive 8-bit pixel format is the default for a three-board configuration. This means that 8 bits are used to assign a pixel value (color map index) to each pixel. This format allows 256 different colors to appear on the screen at one time. The pixels are arranged 1024 x 1024 in visible display memory, and 1024 x 1024 in hidden display. Interactive formats support all GPR operations.

The 8-bit interactive format is compatible with the 4-bit interactive format. For example, the Display Manager uses 4-bit planes, but runs on a configuration using 8-bit planes. In general, the operations performed in an 4-bit format can be performed in 8-bit format.

Optionally, software can change a three-board configuration to a 24-bit imaging format. This means that 24 bits are used to assign a pixel value (color map index) to each pixel, making it possible to use 16 million different colors. The pixels are arranged with 512 x 512 in visible display and 512 x 512 in hidden display. Imaging formats support only limited GPR operations.

Figure 2-2 and Figure 2-3 show the color display formats available for two-board and three-board hardware configurations.

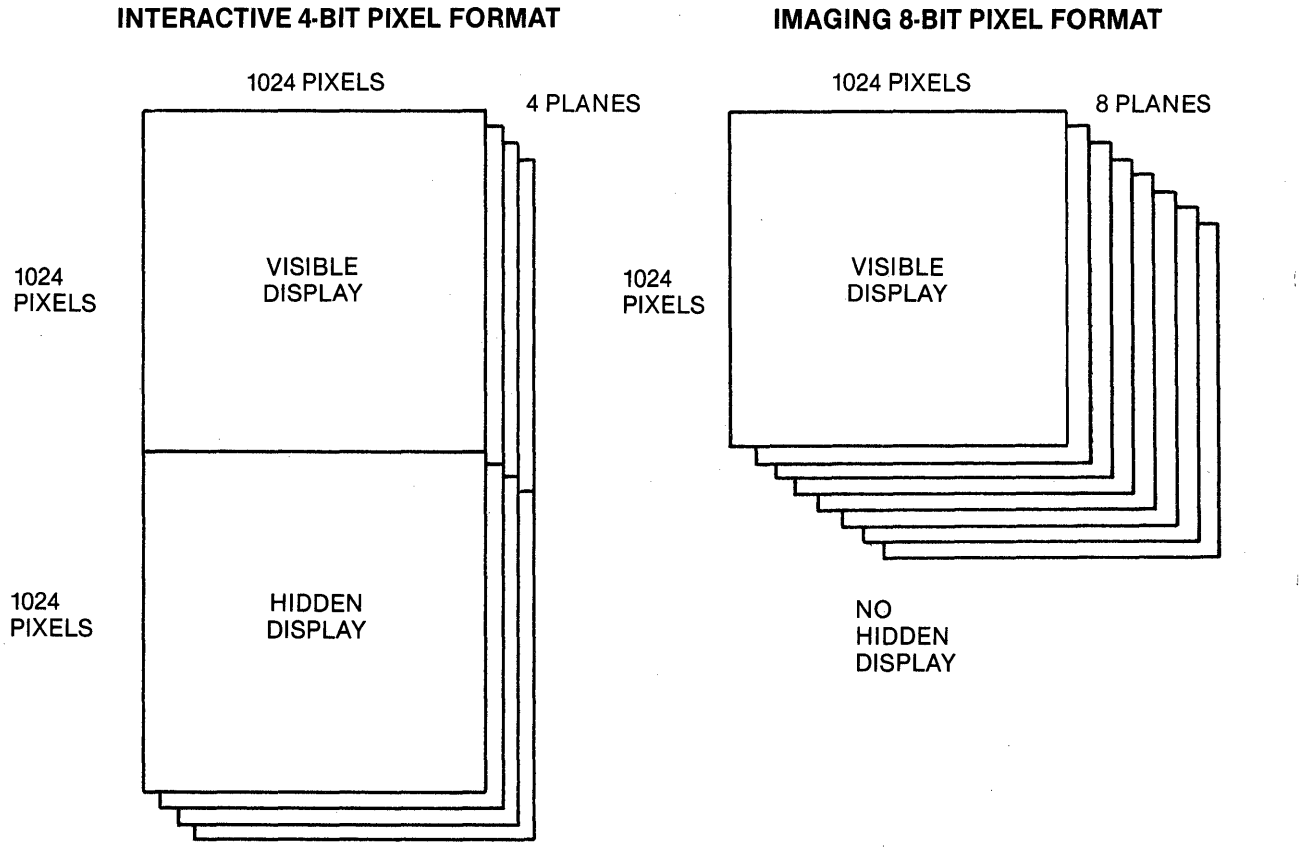


Figure 2-2. Color Display Memory Formats: Two-Board Configuration

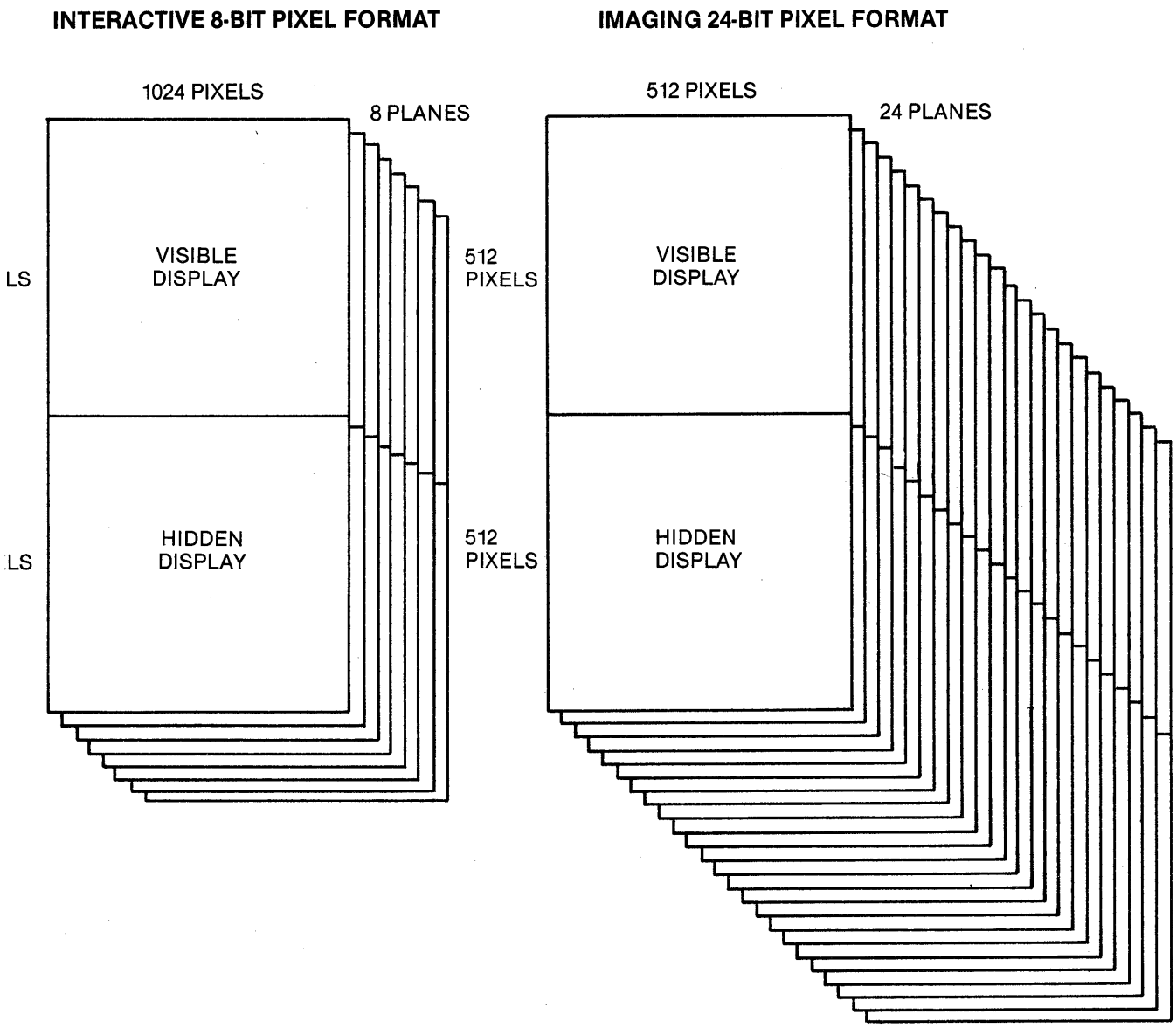


Figure 2-3. Color Display Memory Formats: Three-Board Configuration

2.2 DISPLAY MODES

A graphics program can run in one of four modes:

- Borrow-display mode: on the full screen, which is temporarily borrowed from the Display Manager. This mode has the option not to clear the screen.
- Direct Mode: within a window, which is acquired from the Display Manager for a period of time.
- Frame Mode: within a frame of a Display Manager pad.
- No-Display Mode: without a display, using only bitmaps allocated in main memory.

Programs select a display mode when they initiate a graphics session (with GPR_$INIT). Most of the graphics routines can operate within any of these modes; some routines do not operate in frame mode. Imaging formats require borrow-display mode.

2.2.1 Borrow-Display Mode

In borrow-display mode, the program borrows the full screen and the keyboard from the Display Manager and uses the display driver directly through GPR software. All Display Manager windows disappear from the screen. The Display Manager continues to run during this time. However, it does not write the output of any other processes to the screen or read any keyboard input until the borrowing program returns the display. Input typed ahead into input pads may be read while the display is borrowed. Borrow-display mode is useful for programs that require exclusive use of the entire screen.

An option in borrow-display mode allows you to allocate a bitmap in display memory without setting all the pixels to zero. This is useful for copying what is on the screen into a file to save for later display or printing.

2.2.2 Direct Mode

Direct mode is similar to borrow-display mode, but the program borrows a window from the Display Manager instead of borrowing the entire display. The Display Manager relinquishes control of the window in which the program is executing, but continues to run, writing output and processing keyboard input for other windows on the screen. Direct mode offers a graphics application the performance and unrestricted use of display capabilities found in borrow-display mode and, in addition, permits the application to coexist with other activities on the screen. Direct mode should be the preferred mode for most interactive graphics applications.

Chapter 8 describes how programs perform graphics input and output in direct mode.

2.2.3 Frame Mode

Alternately, a graphics program that executes within a frame of a Display Manager pad calls the Display Manager, which interacts with the display driver. A graphics program executes more slowly in frame mode than in borrow-display mode; however, frame mode offers some additional Display Manager features:

- A frame provides a "virtual display" that can be larger than the window, allowing the user to scroll the window over the frame.
- Frame mode makes it easier to perform ordinary stream I/O to input and transcript pads
- In frame mode, the Display Manager will reproduce the image when necessary.
- The program can leave the image in the pad upon exit so that users can view it at some later time.

Frame mode currently places some restrictions on the GPR operations that are allowed. Chapter 11 describes the individual routines, including their restrictions. One restriction is described below.

Implementation Restrictions on Color/Intensity Values in Frame Mode

Currently, the Display Manager does not store a bitmap when a program operates in frame mode. Therefore, a program using frame mode cannot call graphics routines which depend on particular pixel values of the current bitmap. These routines include pixel read and write operations, BLTs that use source data from the frame, and GPR_$INQ_BITMAP_POINTER.

2.2.4 No-Display Mode

When the program selects no-display at initialization, the GPR initialization routine allocates a bitmap from main memory. The program can then use GPR routines to perform graphic operations to the bitmap, bypassing any screen display entirely. Applications can use no-display mode to create a main memory bitmap, then call graphics map file routines to make a file, or send the bitmap to a peripheral device, such as a printer.

2.3 USING COLOR DISPLAY FORMATS

Interactive display formats fully support all GPR output operations -- block transfer, area filling, line drawing, text manipulation. Imaging display formats support only limited display operations -- displaying (not reading) pixel data and changing the color map (see Chapter 4 for a discussion of color maps). Other functions will return error messages.

Imaging display formats make it possible to display images with more bits per pixel than are available with interactive formats. Additionally, in 24-bit pixel format, select frame operations (GPR_$SELECT_COLOR_FRAME) are allowed. These operations are used to look at either half of display memory.

2.3.1 Using Imaging Display Formats

Switching the display between an interactive format and an imaging format causes the hardware to reconfigure the refresh buffer memory and to rearrange the bitmap. This means that an intelligible image in one format becomes unintelligible in another.

The imaging formats are supported only in borrow-display mode. To change from an interactive to an imaging format, you must be in borrow-display mode.

2.3.2 Routines for Imaging Display Formats

Use the following routines and procedures for imaging display formats. For a detailed description of these calls, see Chapter 11 of this manual.

1. Establish borrow-display mode:

   GPR_$INIT

   You may or may not first want to perform some graphics operations in interactive format.

2. Set the display to 24-bit pixel format:

   GPR_$SET_IMAGING_FORMAT

   Use the format argument to switch to 8- bit or 24-bit imaging format.

3. To inquire about the format, use:

   GPR_$INQ_IMAGING_FORMAT

4. To establish new values for the color map, use:

   GPR_$SET_COLOR_MAP

5. To write pixel data to the display, use:

   GPR_$WRITE_PIXELS

6. To return to interactive format, use the following call with the interactive argument:

   GPR_$SET_IMAGING_FORMAT

7. To terminate the session and return the display to the Display Manager, use:

   GPR_$TERMINATE

CHAPTER 3

MEMORY REPRESENTATION: BITMAPS AND PIXELS

This chapter describes bitmap structure and pixels in relation to graphic storage and representation. The chapter includes discussion of bitmaps in display and main memory, bitmap routines, and bitmap attributes.

A bitmap is a rectangular data structure that stores a graphic image. A pixel (picture element) is a discrete point of a graphic image. Each pixel in a display has a corresponding address in a buffer which stores the image. Thus, pixels provide the means of displaying a graphic image. This section describes bitmap structure and pixels.

3.1 BITMAP STRUCTURE

On a bit-mapped display, a one-to-one correspondence exists between the screen image and display memory. On the screen, a rectangular array (a raster) of pixels forms the visible image. In memory, a bitmap, a rectangular data structure, stores the graphic image.

A bitmap is a three-dimensional array of bits, having width, height, and depth. On the monochromatic display, bitmap width and height may vary; the depth equals one. On the color display, bitmap width, height, and depth can vary.

Bitmap width and height correspond to image width and height. Bitmap width is represented by x-coordinates ranging from zero on the left to a maximum defined for the bitmap on the right. Bitmap height is represented by y-coordinates ranging from 0 on the top to a maximum defined for the bitmap on the bottom (see Figure 3-1). When a program establishes bitmap size, it determines the maximum x- and y-coordinates. For bitmap size limitations, see GPR_$INIT.

Bitmap depth specifies the number of bits of information associated with each pixel. Each one-bit-deep slice of a bitmap is called a plane of the bitmap. This slice has the same width and height as the bitmap.

On the monochromatic display, bitmaps contain one plane; the depth of displayed bitmaps is one bit. On the color display in 4-bit pixel mode, bitmaps can contain up to 4 planes; displayed bitmaps can be up to 4 bits

deep. On the color display in 8-bit pixel mode, bitmaps can contain up to 8 planes; displayed bitmaps can be up to 8 bits deep.

Planes are numbered from zero to the number of bit planes in the bitmap minus one. Plane zero holds the least significant bits of the pixels, and the highest numbered bit plane holds the most significant bits.

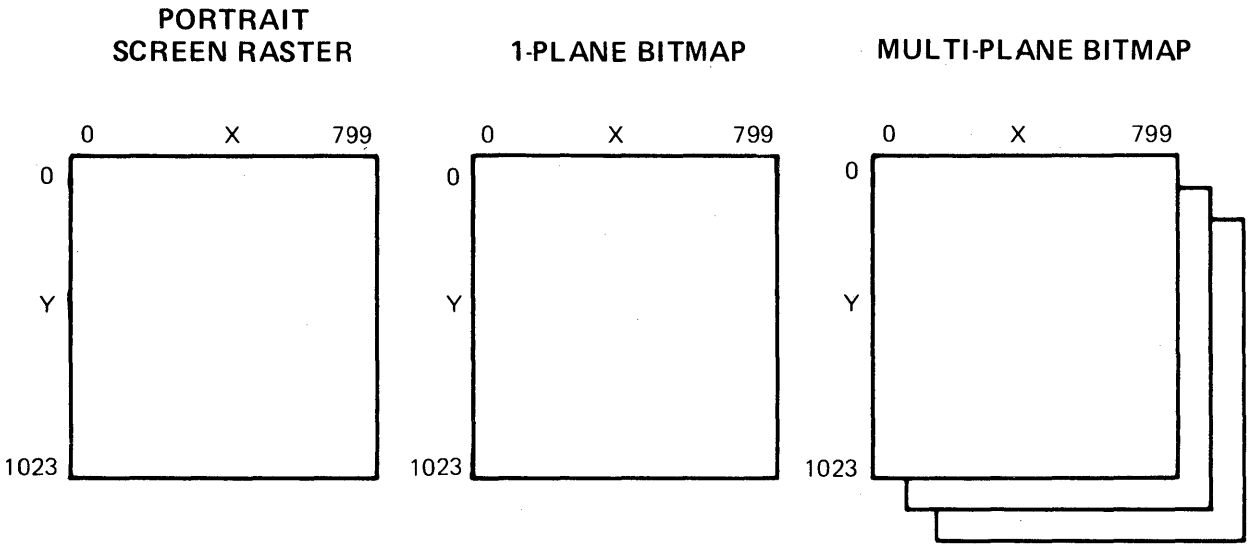


Figure 3-1. Screen and Bitmap Width and Height

3.2 PIXELS

On the monochromatic display, each pixel consists of one bit. On the color display, a bitmap can have multiple planes. Therefore pixels on a color display can consist of a group of bits.

The bitmap pixel values are used by the color map to generate the visible intensity (dark or bright) or color of the corresponding screen pixel. X- and y-coordinates reference pixel positions on a bitmapped (raster) display and in a bitmap.

The distance between adjacent pixels on a raster display is called a raster unit. The coordinate position (0,0) is at the top left corner of the screen, the Display Manager frame, or the bitmap. To change the position of the origin, use GPR_$COORDINATE_ORIGIN.

3.3 BITMAPS IN DISPLAY, MAIN MEMORY, AND EXTERNAL STORAGE

A bitmap may reside in display memory, main memory, or external storage. A single graphics session (beginning with GPR_$INIT and ending with GPR_$TERMINATE) can involve zero, one, or more bitmaps in display memory and multiple bitmaps in main memory. A bitmap resides in only one location at any time.

A bitmap in display memory is visible on the screen. A bitmap in main memory must be copied to display memory to become visible. A bitmap file may be opened to create a bitmap and store it on the disk. The bitmap on which a program is operating, whether it resides in display or main memory, is called the current bitmap. The first bitmap created in a graphics session is called the initial bitmap.

3.3.1 Bitmaps in Display Memory

Most programs that include a bitmap in display memory can execute in borrow-display mode, direct mode, or frame mode. When a program initializes the graphics primitives, it may select any of these modes. Programs that use the imaging format for color display must be initialized in borrow-display mode. These display modes pertain only to programs which use the screen for display of images. For bitmaps of programs that do not display an image, see the section below.

On a monochromatic display, bitmaps in display memory can contain only one plane. The initial maximum bitmap size is the size of the visible monochromatic display, 800x1024 pixels for portrait displays, and 1024x800 for landscape displays.

For a color display in a 4-bit interactive format, display memory bitmaps can contain up to four planes; in 8-bit interactive or imaging formats, they can contain up to eight planes. In 24-bit imaging format, they can contain up to 24 planes. The initial maximum bitmap size is the size of the visible color display: 1024x1024 pixels for 4-bit and 8-bit formats; 512 x 512 for a 24-bit format.

3.3.2 Bitmaps in Main Memory

If a program selects not to display an image on the screen, it can manipulate the image by allocating a bitmap in main memory. A program might require main memory bitmaps, for example, to operate on a bitmap larger than that available in display memory.

Main memory bitmaps can contain up to eight planes, arranged 4096x4096 pixels, regardless of the display in use. However, if a main memory bitmap is transferred to display memory, the size of the bitmap transferred is restricted according to the display type.

3.3.3 Bitmaps in External Storage

The routine GPR_$OPEN_BITMAP_FILE allows you to create or open a file for storage of a bitmap on the disk. This means that you can define a graphic image on a bitmap, store the bitmap in a file, and then access it for display at a later time. You can treat bitmaps for external storage like any other GPR bitmap.

3.3.4 Initial Bitmap

When a program calls GPR_$INIT to initialize the graphics package, it selects a mode of operation and initial bitmap size. An initial bitmap is created as follows:

- If the program uses the display and operates in frame mode, the initial (and current) bitmap is displayed and is the same size and location as the frame. The graphics primitives create the frame with the size parameters which the program specifies.

- If the program operates in direct or borrow-display mode, the initial (and current) bitmap is also displayed. The program chooses the bitmap size, which can be equal to or less than the size of the screen. If the bitmap is smaller than the screen, it is located in the top left-most area of the screen or window. The (0,0) bitmap coordinate is in the same position as the (0,0) screen coordinate.

- If the program does not use the display, the graphics primitives initialization routine designates an initial bitmap in main memory. This bitmap has the dimensions which the program has selected. In this case the initial, current bitmap is not displayed.

3.4 ROUTINES FOR CREATING, CANCELING, IDENTIFYING BITMAPS

The following routines include features for creating, deleting, and identifying bitmaps:

- GPR_$INIT -- initializes the graphics package and establishes a current bitmap either in display memory or main memory, as specified by the program.

- GPR_$TERMINATE -- terminates the graphics package and its bitmaps.

- GPR_$ALLOCATE_BITMAP -- allocates a bitmap in main memory and returns a unique descriptor for the bitmap.

- GPR_$DEALLOCATE_BITMAP -- deallocates an allocated bitmap.

- GPR_$ALLOCATE_BITMAP_NC -- allocates a bitmap in main memory without setting all the pixels in the bitmap to zero, and returns a unique descriptor for the bitmap.

- GPR_$ALLOCATE_HDM_BITMAP -- allocates a bitmap in hidden display memory.

- GPR_$SET_BITMAP -- establishes a specified bitmap, in display memory or main memory, as the current bitmap.

- GPR_$SET_WINDOW_ID -- establishes the character that identifies the current bitmap's window.

- GPR_$INQ_WINDOW_ID -- returns the character that identifies the current bitmap's window.

- GPR_$INQ_BITMAP -- returns the unique descriptor for the current bitmap, which is in either display memory or main memory.

- GPR_$INQ_BITMAP_DIMENSIONS -- returns the size and number of planes of a bitmap.

- GPR_$SET_BITMAP_DIMENSIONS -- changes the size of a previously created bitmap.

- GPR_$INQ_BM_BIT_OFFSET -- returns the offset of the left edge of the bitmap in virtual address space.

- GPR_$OPEN_BITMAP_FILE -- Opens a file for external storage of a bitmap.

3.5 ACCESSING AND MANIPULATING BITS IN A BITMAP

As described above, a bitmap is a data structure. When a program allocates a bitmap, the graphics package maps this data structure into the virtual address space. The program has no control of the location or layout of a bitmap in virtual address space. Typically, the bits constituting one bitmap scan line (one horizontal line in a single plane) occupy sequential locations in the address space. Next, a series of address locations is skipped. Then, all the bits of the second scan line are stored, and so on. For a given bitmap, the offset in memory from the beginning of one scan line to the next remains constant.

For multiple-plane bitmaps in main memory, plane 1 starts on the scan line immediately following the last scan line of plane 0. For bitmaps stored in bitmap files, this is not necessarily true. The routine GPR_$OPEN_BITMAP_FILE returns the offset in memory between planes. Unlike main memory bitmaps, displayed multiple-plane bitmaps on the color display can be accessed only one plane at a time. Programs must call the routine GPR_$REMAP_COLOR_MEMORY to select the plane to access.

To access a bit location in order to clear or set the bit value, a program does not specify bitmap x- and y-coordinates. The program must find the location of the bit in the virtual address space to manipulate it. To find bit locations, programs can use the routine GPR_$INQ_BITMAP_POINTER. This routine returns a pointer to the beginning address of the bitmap's storage, as well as the number of 16-bit words (the offset) between the beginning of storage for each successive scan line.

If a program uses the pointer routine to get the address of display memory (monochromatic and color), it should call the routine GPR_$ENABLE_DIRECT_ACCESS after any calls that change the display and before using the pointer returned by GPR_$INQ_BITMAP_POINTER. This ensures that any pending display hardware operations are complete before the program uses the pointer to access display memory (with the routines GPR_$READ_PIXELS and GPR_$WRITE_PIXELS).

For example, to set a bit to 1 at the bitmap coordinate position x = 64, y = 3, you can write a section of a FORTRAN program, using the pointer statement, as follows:

```
      INTEGER*2 X, Y
      INTEGER*2 BIT_WORD
      INTEGER*4 INDEX
C
      INTEGER*4 BITMAP_PTR, BITMAP_DESCRIPTOR, STATUS
      INTEGER*2 OFFSET
C
      INTEGER*2 BITMAP
      POINTER /BITMAP_PTR/ BITMAP(0:65535)
      .
      .
      .
      CALL GPR_$ENABLE_DIRECT_ACCESS(STATUS)
      CALL GPR_$INQ_BITMAP_POINTER(BITMAP_DESCRIPTOR, BITMAP_PTR,
     1                             OFFSET, STATUS)
C
C     To set bit at (x = 64, y = 3) to 1:
C
      X = 64
      Y = 3
      INDEX = (Y * OFFSET) + (X / 16)
      BIT_WORD = RSHFT(16#8000, MOD(X, 16))
      BITMAP(INDEX) = OR(BITMAP(INDEX), BIT_WORD)
```

3.6 MULTIPLE DISPLAYED BITMAPS

To initialize more than one window with graphics primitives, you can use GPR_$INIT more than once. To do so, you must specify frame or direct display mode in the unit parameter. The stream identifier for the pad must be different for each window.

You can draw in all the windows, one at a time, by calling GPR_$SET_BITMAP as for other drawing operations. The bitmap descriptor is returned automatically for each initialized window.

Calls which access the display bitmap go to the most recently current screen bitmap. This may be the current screen bitmap or the screen bitmap that was most recently current if the current bitmap is a memory bitmap.

With multiple displays, input events can come from several different bitmaps. To identify the source of an input event, the routine GPR_$EVENT_WAIT or GPR_$COND_EVENT_WAIT returns a character. This character is established with the routine GPR_$SET_WINDOW_ID, associated with an entered window event. That character tells you which window you have just entered. The source of input remains the same until a left window event is received.

To establish the identifying character for a window, use the routine GPR_$SET_WINDOW_ID. This routine accepts a character which identifies the current screen bitmap or the most recent screen bitmap. If the current bitmap is the screen bitmap, it references that one. If the current bitmap is in memory, the routine uses the most recently current bitmap.

Use GPR_$SET_WINDOW_ID to associate a different character with each window. A program must enable entered window events for all windows that will have such events.

3.7 BITMAP ATTRIBUTES

Each bitmap has a set of attributes which specify the manner in which subsequent operations on the bitmap will be performed. For example, attributes can specify that only a certain section of the bitmap can be manipulated in any subsequent operations or that text written on the bitmap will be displayed in specified fonts.

3.7.1 Description of Attributes

For each bitmap, the program must allocate an attribute block and then assign it to the bitmap. The attributes in the block specify characteristics of graphics operations.

Bitmap attributes are the following: clipping window, coordinate origin, draw value (pixel value), fill value (pixel value), text value, text background value, text font, line style, plane mask, and raster operation. These attributes are described below.

Clipping Window

The clipping window attribute specifies a rectangular section of the bitmap, outside which no pixels can be modified, as suggested in Figure 3-2. (GPR_$SET_CLIPPING_ACTIVE does not draw a rectangle, but specifies a rectangular area.) After a program calls the routine GPR_$SET_CLIP_WINDOW to specify the clipping window, it may call GPR_$SET_CLIPPING_ACTIVE to enable the clipping window.

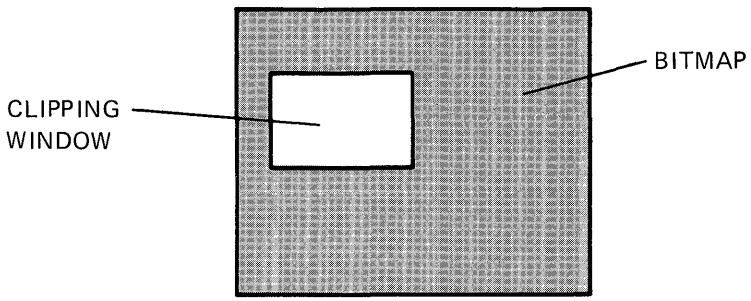


Figure 3-2. Clipping Window on a Bitmap

Coordinate Origin

The coordinate origin specifies a pair of offset values to add to all coordinate positions. These values are subsequently used as input to move, line drawing, or bit block transfer (BLT) operations on the current bitmap. For example, the coordinate origin affects calls to the routines GPR_$MOVE, GPR_$LINE, and GPR_$PIXEL_BLT.

Draw Value

The draw pixel value specifies the value to which pixels will be set when drawing lines.

Fill Value

The fill pixel value specifies the value to which pixels will be set when filling areas.

Text Value

The text pixel value specifies the value to which pixels will be set to write text.

Text Background Value

The text background pixel specifies the value to which pixels will be set for text background.

Text Font

The text font attribute specifies the font in which to display text characters in the bitmap.

Line Style

The line style attribute specifies the style in which to display line segments in the bitmap. Line style can be either solid or dashed; if dashed, the style scale factor determines the length of the dash.

Plane Mask

The plane mask specifies which planes of a bitmap can be modified by any graphics operation and which planes are protected from modification.

Raster Operation

A raster operation specifies the manner in which to combine pixels in one plane of source and destination bitmaps to form a new destination bitmap. The value of each new destination bit is assigned by a Boolean function of the previous value of each destination bit and the value of the corresponding source bit.

Sixteen raster operations form the set of rules for combining bit values. Assigning a raster operation code to a plane of a bitmap alters no values. The raster operation code controls how values are logically combined when a program subsequently draws or uses a bit block transfer (BLT) to combine two bitmaps. (See Chapter 6 for a description of BLTs and their use of raster operations.) Table 3-1 lists the op codes and logical functions for the sixteen raster operations. Table 3-2 is a truth table of the raster operations.

Table 3-1. Raster Operations and Their Functions

Op Code	Logical Function
0	Assign zero to all new destination values.
1	Assign source AND destination to new destination.
2	Assign source AND complement of destination to new destination.
3	Assign all source values to new destination. (Default)
4	Assign complement of source AND destination to new destination.
5	Assign all destination values to new destination.
6	Assign source EXCLUSIVE OR destination to new destination.
7	Assign source OR destination to new destination.
8	Assign complement of source AND complement of destination to new destination.
9	Assign source EQUIVALENCE destination to new destination.
10	Assign complement of destination to new destination.
11	Assign source OR complement of destination to new destination.
12	Assign complement of source to new destination.
13	Assign complement of source OR destination to new destination.
14	Assign complement of source OR complement of destination to new destination.
15	Assign one to all new destination values.

Table 3-2. Raster Operations: Truth Table

SOURCE BIT VALUE	DESTINATION BIT VALUE	RESULTANT BIT VALUES FOR THE FOLLOWING OP CODES: 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

3.7.2 Establishing and Changing Attributes

An attribute block is required to establish bit map attributes. The initial bitmap has an attribute block associated with it. A program must associate an attribute block with each subsequent bitmap, using the following procedure:.

- The program first calls GPR_$ALLOCATE_ATTRIBUTE_BLOCK. This routine allocates an attribute block containing a set of default attributes, and returns a unique descriptor of the attribute block.
- Next the program must call GPR_$SET_ATTRIBUTE_BLOCK with the attribute descriptor to associate the attribute block with the current bitmap.

A program can associate an attribute block with more than one bitmap. However, if the program changes any attributes in the current bitmap, that attribute will be changed on all other bitmaps which have the same attribute block as the current bitmap.

The default attribute settings are shown in Table 3-3.

Table 3-3. Default Attribute Settings

Attribute	Default Setting
Clipping Window	Same size as bitmap. If the program reassigns the attribute block from one bitmap to a smaller bitmap, the clipping window is automatically reduced to the new bitmap size.
Clipping Active	For borrow-display and frame mode, false; clipping window disabled. For direct mode, true; clipping window enabled.
Coordinate Origin	(0,0)
Draw Value	1
Fill Value	1
Text Value	1, for borrowed displays, memory bitmaps, and display manager frames on monochromatic displays; 0, for Display Manager frames on color displays.
Text Background Value	-2 (same as bitmap background, which is 0 for borrowed displays and memory bitmaps, and the same as the window background for display manager frames).
Font	No default. Program must load and set font.
Line Style	Solid line.
Plane Mask	All planes can be modified.
Raster Op	Op = 3, set all destination bit values to source bit values.

To change an individual attribute associated with the current bitmap, the program calls one of the attribute-setting routines which acts directly on the bitmap. These include, for example, GPR_$SET_CLIP_WINDOW, GPR_$SET_RASTER_OP, and GPR_$SET_TEXT_FONT. When changing attributes, the program does not need to change the attribute block.

The routines for setting and retrieving attributes are listed below:

Set Attributes

- GPR_$SET_CLIP_WINDOW -- changes the clipping window for the current bitmap.
- GPR_$SET_CLIPPING_ACTIVE --enables/disables a clipping window for the current bitmap.
- GPR_$SET_COORDINATE_ORIGIN --establishes x- and y-offsets to add to all x- and y-coordinates used as input for these operations: moving the current position, line drawing, and block transfers.
- GPR_$SET_DRAW_VALUE -- specifies the color/intensity to use to draw lines.
- GPR_$SET_FILL_BACKGROUND_VALUE -- specifies the color/intensity value used for drawing the background of tile fills.
- GPR_$SET_FILL_PATTERN -- specifies the fill pattern to use for the current bitmap.
- GPR_$SET_FILL_VALUE --specifies the color/intensity to use to fill rectangles.
- GPR_$SET_LINESTYLE -- specifies the linestyle as solid or dashed.
- GPR_$SET_LINE_PATTERN -- establishes the pattern used in drawing lines.
- GPR_$SET_PLANE_MASK -- establishes a plane mask that specifies which planes to use for subsequent write operations.
- GPR_$SET_RASTER_OP -- specifies a new raster operation for BLTs and lines.
- GPR_$SET_TEXT_BACKGROUND_VALUE -- specifies the color/intensity to use for text background.
- GPR_$SET_TEXT_FONT -- establishes a new font for subsequent text operations.
- GPR_$SET_TEXT_VALUE -- specifies the color/intensity to use for writing text.

Retrieve Attributes

- GPR_$INQ_CONSTRAINTS -- returns the clipping window and plane mask used for the current bitmap.
- GPR_$INQ_COORDINATE_ORIGIN -- returns the x- and y-offsets added to all x- and y-coordinates used as input to move, line drawing, and BLT operations on the current bitmap.
- GPR_$INQ_DRAW_VALUE -- returns the color/intensity value used for drawing lines.
- GPR_$INQ_FILL_BACKGROUND_VALUE -- returns the color/intensity value used for drawing the background of tile fills.
- GPR_$INQ_FILL_PATTERN -- returns the fill pattern in use for the current bitmap.
- GPR_$INQ_FILL_VALUE -- returns the color/intensity value used for filling rectangles.
- GPR_$INQ_LINE_PATTERN -- returns the pattern used in drawing lines.
- GPR_$INQ_LINESTYLE -- returns information about the current linestyle.
- GPR_$INQ_RASTER_OPS -- returns the raster operations in use for the current bitmap.
- GPR_$INQ_TEXT -- returns the text font and text path used for the current bitmap.
- GPR_$INQ_TEXT_OFFSET -- returns the x- and y-offsets from the top left pixel of a string to the origin of the string's first character. This routine also returns the pixel which is the new current position after the text is written with GPR_$TEXT.
- GPR_$INQ_TEXT_VALUES -- returns the current values of color/intensity for text and text background in the current bitmap.

CHAPTER 4

COLOR/INTENSITY SPECIFICATION

This chapter describes the components and uses of a color map. The distinctive characteristics of a color map in interactive and imaging formats are presented.

4.1 THE COLOR MAP: A SET OF COLOR VALUES

A color map is a set of color values, each representing a visible color or intensity. A color value is an encoding of a particular visible color/intensity, based on the RGB (red/green/blue) color model. The RGB color model defines red, green, and blue as primary colors. All other colors are combinations of these primaries, including the three secondary colors (cyan, magenta, and yellow).

Graphics programs use a color map to specify color and intensity (gray-scale) values. A program can redefine the color map to assign colors to pixel values. To assign different colors to lines or other graphic entities, a program must draw them using different pixel values and then assign the appropriate colors to these pixel values.

Each color value is 24 bits in length and is divided into three 8-bit fields. The first field stores the value for the red component of the color, the second for the green component, and the third for the blue component. Each field can have a value in the range 0-255. The 24-bit structure allows for approximately 16 million different color values.

A field value of zero specifies the absence of the primary color, and a field value of 255 specifies full intensity of that primary color. Figure 4-1 illustrates the structure of a color value. (Color values are declared as 4-byte integers; the value of the most significant byte is ignored.)

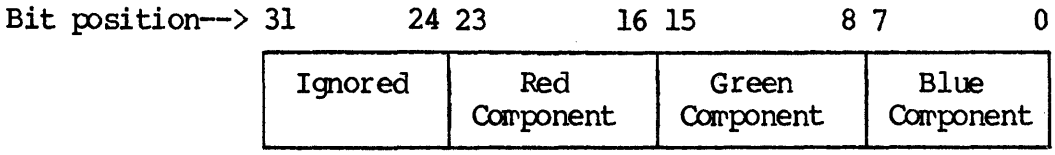


Figure 4-1. Color Value Structure

If all fields have equal values, the color value is a shade of gray, as Table 4-1 shows.

Table 4-1. Example of Gray-Scale Color Values and Visible Intensities

Color Value			Visible Color/Intensity Represented by the Color Value
R field	G field	B field	
255	255	255	white
191	191	191	light gray
127	127	127	medium gray
63	63	63	dark gray
0	0	0	black

4.2 ESTABLISHING A COLOR MAP

A color map consists of a set of color map entries -- each is a color value associated with an index (see Table 4-3). Though the association between color values and visible colors/intensities cannot be changed, a program can establish and change the association between indexes and color values by changing the entries in the color map. In this way, a program can select the set of colors/intensities to constitute a color map for a particular application, and associate them with particular indexes. See the routines GPR_$SET_COLOR_MAP and GPR_$INQ_COLOR_MAP.

At initialization of the graphics primitives package, a default color map is established. The default color maps for monochromatic and color displays are described below. In frame mode or direct mode, a program cannot modify color map entries 0 and 7-15.

4.3 USING A COLOR MAP

After a color map is established, a program can use it to specify the color/intensity to use for displaying lines, text, text background, rectangles, and the full screen, as follows. The program assigns a pixel value (color map index) to the draw value attribute, the text value attribute, the text background value attribute, the fill value attribute, and/or uses the index to clear the screen. See the description of the following routines:

```
GPR_$SET_DRAW_VALUE
GPR_$INQ_DRAW_VALUE
GPR_$SET_TEXT_VALUE
GPR_$SET_TEXT_BACKGROUND_VALUE
GPR_$INQ_TEXT_VALUES
GPR_$SET_FILL_VALUE
GPR_$INQ_FILL_VALUE
GPR_$CLEAR
```

4.3.1 Color Map for Monochromatic Displays

For a monochromatic display, the color map has only two entries. The default color map assigns the color value 0 to color map index 0, and the color value 1 to color map index 1, as Table 4-2 shows. If a program uses the default color map and sets a particular bitmap pixel to 1 (GPR_$WHITE), the corresponding pixel on the screen appears bright. If it selects 0 (GPR_$BLACK), the corresponding pixel would appear dark.

Table 4-2. Default Color Map for Monochromatic Displays

Color Table Index	Color Value	Resultant Visible Color/Intensity
0	0 (GPR_$BLACK)	black
1	16#FFFFFF (GPR_$WHITE)	white

4.3.2 Color Map for Color Displays: 4-Bit and 8-Bit Formats

For a color display in the 4-bit pixel format, the color map has 16 entries, with index values 0-15. For a color display in the 8-bit pixel format, the color map has 256 entries, with index values 0-255. In both formats, all entries are set to default values at the initialization of the graphics primitives package. Table 4-3 shows the default color map for color displays.

Table 4-3. Default Color Map for Color Displays

Color Table Index	Color Value				Resultant Visible Color/Intensity
	R	G	B		
0	0	0	0	(GPR_$BLACK)	black
1	255	0	0	(GPR_$RED)	red
2	0	255	0	(GPR_$GREEN)	green
3	0	0	255	(GPR_$BLUE)	blue
4	0	255	255	(GPR_$CYAN)	cyan
5	255	255	0	(GPR_$YELLOW)	yellow
6	255	0	255	(GPR_$MAGENTA)	magenta
7	255	255	255	(GPR_$WHITE)	white
8-15	contain colors used by the Display Manager to display windows.				
16-255	0	0	0	(GPR_$BLACK)	black

4.3.3 Color Map for Color Displays: 24-Bit Format

The color map in 24-bit format functions differently from the color map for other formats. In a 4- or 8-bit format, a single color table index is used to look up one slot in the table. This slot is 24-bits wide: 8 bits each for red, green, and blue. These three taken together describe the color.

In 24-bit format, the color map for the 24-bit pixel format can be described as divided vertically into three parts, each with 8 bits in and 8 bits out. Three independent color table indices are used, each to look up on 8-bit color component value. This means that the red piece of the 24-bit value is looked up in the red column and is referenced by its own index. The green piece is looked up in its column and referenced by its own index. The same is true for blue.

The call GPR_$SET_COLOR_MAP uses the same parameters for 24-bit pixels as it does for 8-bit pixels. The difference is in the number of colors that can be displayed with 24-bit pixels. The primary application for this extended color range is color correction for true-color imaging.

Figures 4-2 and 4-3 illustrate the differences in the two types of color maps.

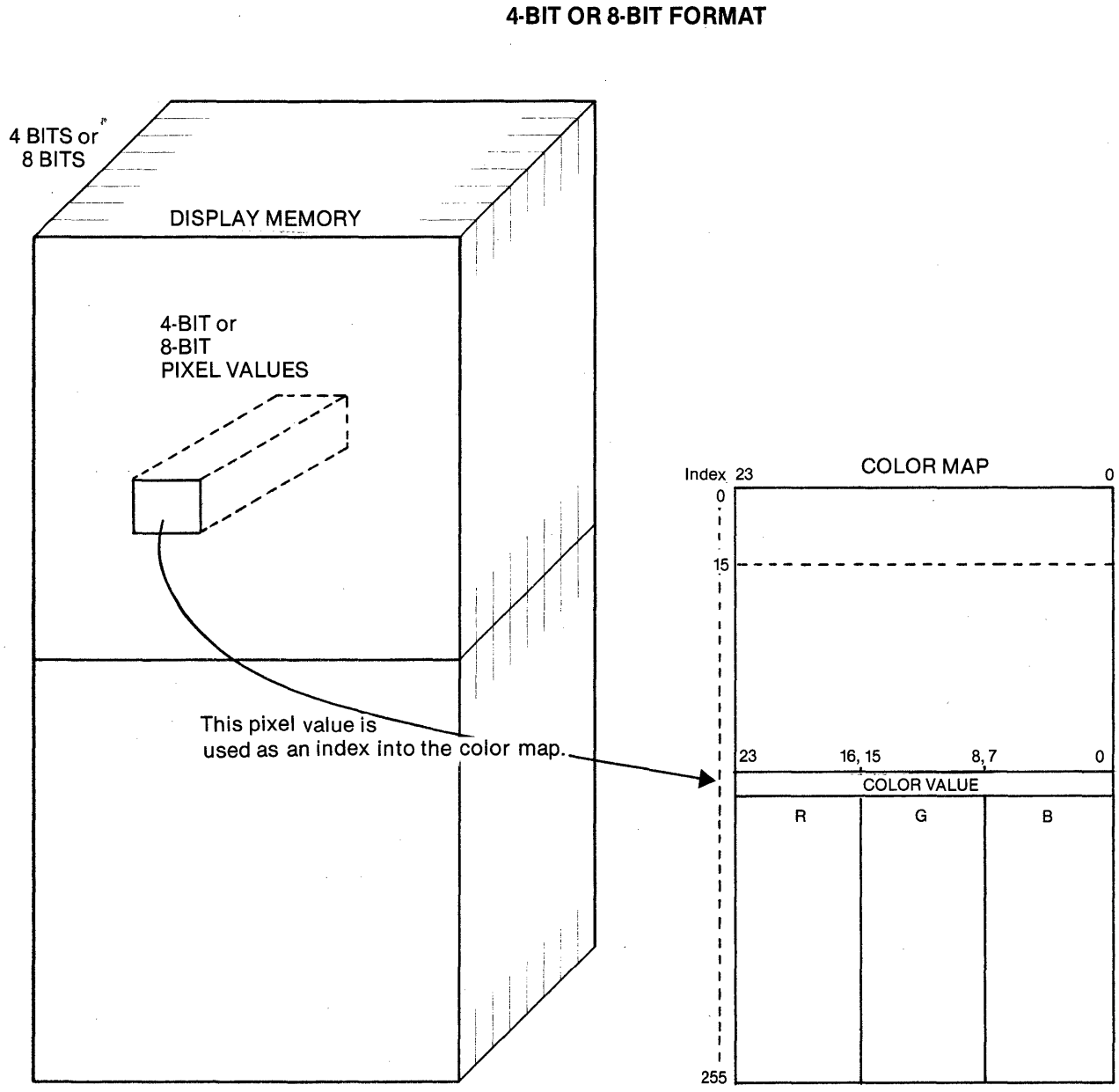


Figure 4-2. 8-Bit Color Map

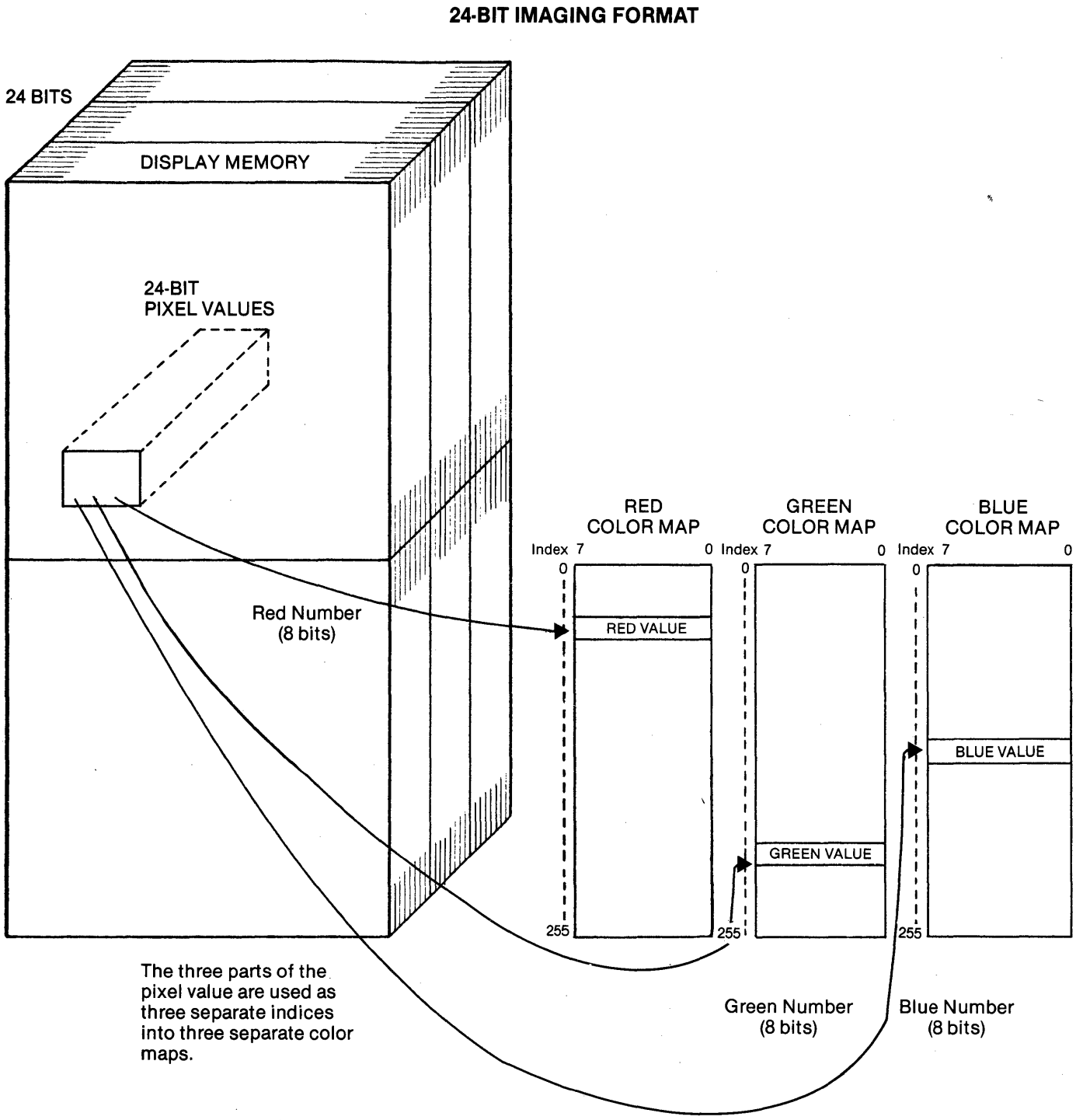


Figure 4-3. 24-Bit Color Map

4.3.4 Saving/Restoring Pixel Values

In interactive formats, a program can read the pixel values of each pixel in a bitmap or section of a bitmap and store the values in a pixel array. Imaging formats do not permit read operations. In both interactive and imaging formats, a program can write the pixel values from a pixel array into a bitmap. See the routines GPR_$READ_PIXELS and GPR_$WRITE_PIXELS.

CHAPTER 5

DRAWING AND TEXT OPERATIONS

This chapter explains the uses of current position and describes drawing, filling, and text operations.

5.1 CURRENT POSITION, CHANGE POSITION

Drawing and text operations within a bitmap always begin at the "current position". Initially, the current position is set to the coordinate position at the top left corner of the bitmap (0,0). To change the current position, use the routine GPR_$MOVE. GPR_$MOVE simply changes the current position, it does not draw any lines, as Figure 5-1 shows.

```
C  X contains 400
C  Y contains 400
   .
   .
   .
   CALL GPR_$MOVE (X,Y,status)
   .
   .
   .
```

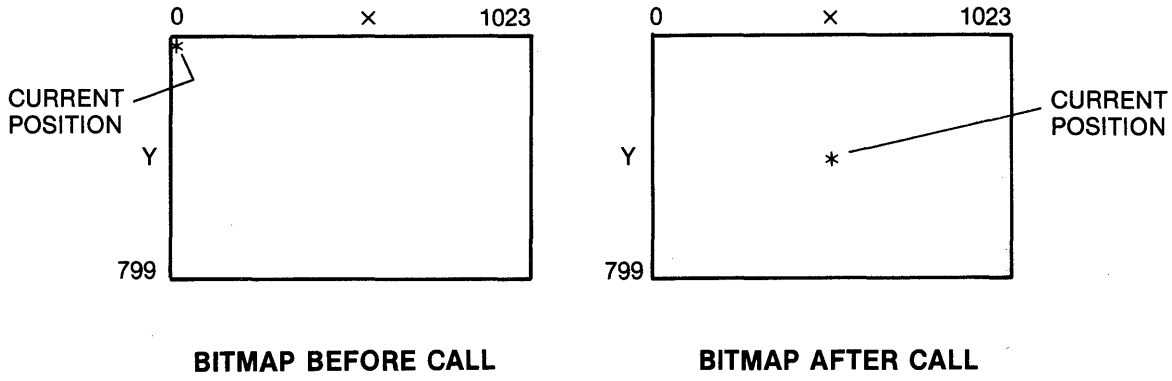


Figure 5-1. Current Position and Changed Current Position

5.2 DRAWING AND FILLING OPERATIONS

The graphics primitives package includes routines that draw lines and fill in circles, rectangles, triangles, trapezoids, and polygons. The circle, rectangle, triangle, and trapezoid routines fill in a specified circle, rectangle, triangle, trapezoid or list of trapezoids. The polygon routines define a polygon, draw and fill it immediately, or save its definition for later drawing and filling. An arc routine and three spline routines draw lines through specified points.

The draw-line routines draw lines from the current position to the new position, and update the current position to the new position. The graphics primitives compute the pixel values of pixels along a line as follows. For each pixel included in the line, the current draw value combines with the pixel's current value, using the raster operations in effect.

The following FORTRAN example and Figure 5-2 show how the graphics primitives compute the pixel value as a line is drawn.

```
C
C  Set raster op to do exclusive OR on plane zero.
C
   CALL GPR_$SET_RASTER_OP (0, 6, STATUS)
C
C  Set the line drawing value to 1.
C
   CALL GPR_$SET_DRAW_VALUE (1, STATUS)
C
C  Draw a line.
C
   CALL GPR_$MOVE (4, 0, STATUS)
   CALL GPR_$LINE (7, 11, STATUS)
```

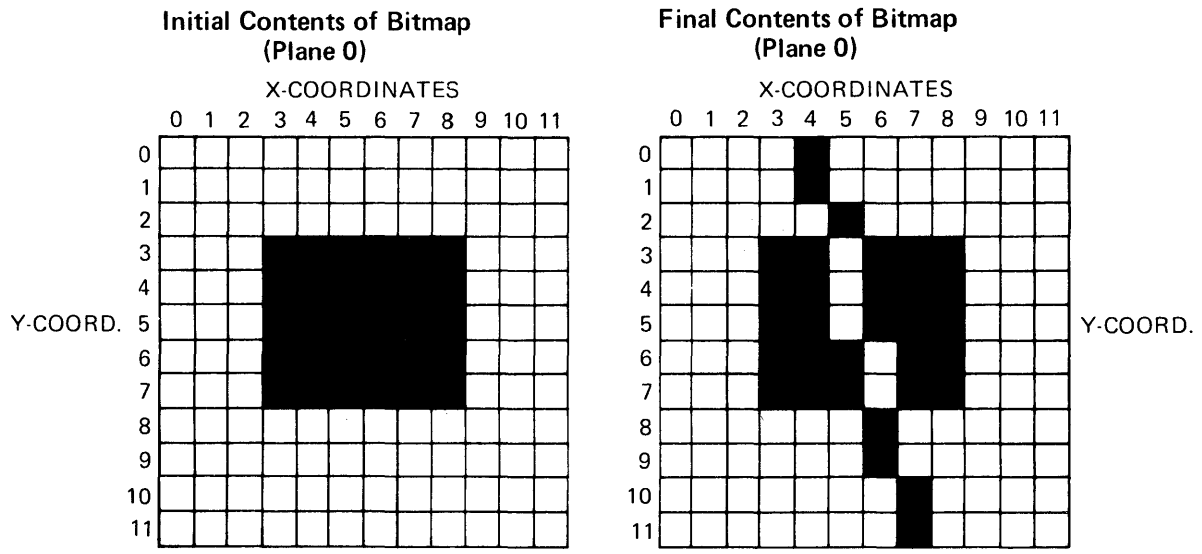


Figure 5-2. Example of Color Value Computation for Line Drawing

The rectangle, triangle, and trapezoid routines fill in a specified rectangle, triangle, trapezoid or list of trapezoids. The rectangle routine fills a rectangle by writing the current fill value into the rectangle without regard to its previous contents or the raster operations in effect. The triangle, trapezoid, and multitrapezoid routines compute the current fill value in the same way as the rectangle routine.

The polygon routines open and define the boundaries of a polygon, and either close and fill the polygon immediately, or close the polygon and return its definition to the program for later drawing and filling. The routine GPR_$PGON_POLYLINE does not draw a polygon; the routine defines a series of line segments for decomposition into trapezoids for filling operations.

A polygon's boundary consists of one or more closed loops of edges. The polygon routine GPR_$START_PGON establishes the starting point for a new loop, closing off the old loop if necessary. The polygon routine GPR_$PGON_POLYLINE defines a series of edges in the current loop.

The polygon routines GPR_$CLOSE_FILL_PGON and GPR_$CLOSE_RETURN_PGON close a polygon by decomposing it into trapezoids. The graphics primitives define a trapezoid as a quadrilateral with two horizontally parallel sides. The polygon routines examine the polygon and break it into trapezoids that can be filled immediately or returned in an array to the program. At a later time, the program can reconstruct the polygon by filling the saved trapezoids with the multitrapezoid routine.

The polygon routines define the interior of a polygon to be all points from which a semi-infinite line will cross the polygon boundary an odd number of times. The graphics primitives fill polygon interiors with the current fill value regardless of previous contents.

The line drawing and rectangle filling routines are shown in the following FORTRAN examples.

The GPR_$LINE routine draws one line, as shown in Figure 5-3.

```
C  Current position is (400,400).
C  X contains 600.
C  Y contains 200.
   .
   .
   .
   CALL GPR_$LINE (X,Y,status)
   .
   .
   .
```

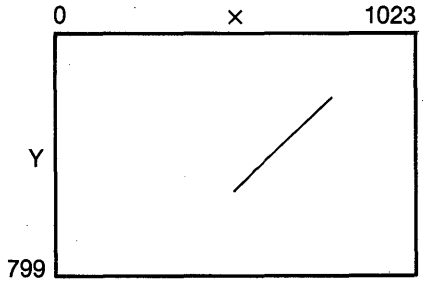


Figure 5-3. Line Drawing Example

The GPR_$POLYLINE routine draws a sequence of connected lines, as Figure 5-4 shows.

```
C  Current position is (100,100).
C  XARRAY contains values 100, 400, 400, 600.
C  YARRAY contains values 200, 100, 500, 200.
C  NPOSITIONS = 4.
     .
     .
     .
   CALL GPR_$POLYLINE (XARRAY, YARRAY, NPOSITIONS, STATUS)
     .
     .
     .
```

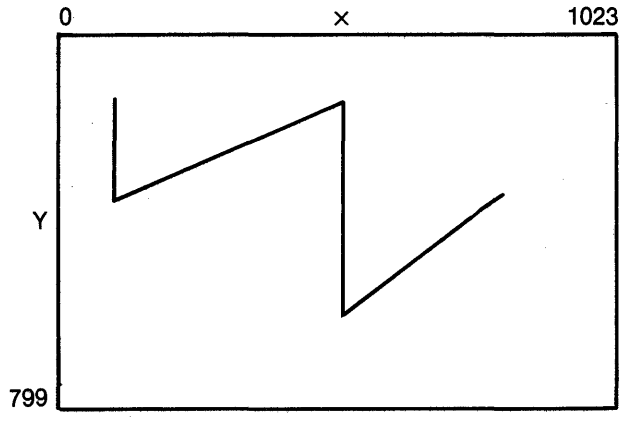


Figure 5-4. Polyline Drawing Example

The GPR_$MULTILINE routine draws a sequence of disconnected lines (alternating moves and draws), as Figure 5-5 shows.

```
C  XARRAY contains values 100, 400, 400, 600.
C  YARRAY contains values 200, 100, 500, 200.
C  NPOSITIONS = 4.
   .
   .
   .
   CALL GPR_$MULTILINE (XARRAY, YARRAY, NPOSITIONS, STATUS)
   .
   .
   .
```

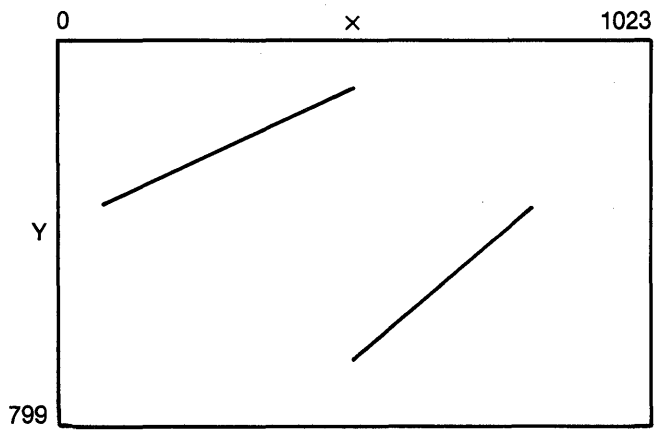


Figure 5-5. Multiline Drawing Example

The GPR_$RECTANGLE routine fills in a rectangle, as Figure 5-6 shows.

```
C  The rectangle is specified by a data item of the type
C       GPR_$WINDOW_T.
C  WINDOW contains values 100, 200, 300, 100.
   .
   .
   .
   CALL GPR_$RECTANGLE (WINDOW, STATUS)
   .
   .
   .
```

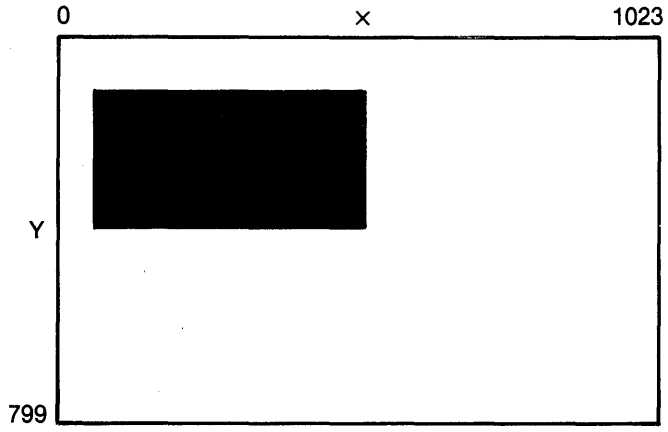


Figure 5-6. Rectangle Fill Operation

The following routines draw lines and polygons and fill areas.

Line Drawing Operations

- GPR_$ARC_3P -- draws an arc from the current position, through two other points.
- GPR_$CIRCLE -- draws a circle with a specified radius around a specified center point.
- GPR_$DRAW_BOX — draws an unfilled box given two opposing corners.

- GPR_$LINE -- draws a line from the current position to the given position. The routine then updates the current position to the given position.
- GPR_$MULTILINE -- draws a series of disconnected lines.
- GPR_$POLYLINE -- draws a series of connected lines.
- GPR_$SPLINE_CUBIC_P -- draws a parametric cubic spline through the control points.
- GPR_$SPLINE_CUBIC_X -- draws a cubic spline as a function of x through the control points.
- GPR_$SPLINE_CUBIC_Y -- draws a cubic spline as a function of y through the control points.

Fill Operations

- GPR_$CIRCLE_FILLED -- draws a solid circle around the center point with the given radius.
- GPR_$RECTANGLE -- fills a rectangle.
- GPR_$TRIANGLE -- fills a triangle.
- GPR_$TRAPEZOID -- draws and fills a trapezoid.
- GPR_$MULTITRAPEZOID -- draws and fills a list of trapezoids.
- GPR_$START_PGON -- defines the starting position to create a loop of edges for a polygon boundary.
- GPR_$PGON_POLYLINE -- defines a series of line segments forming part of a polygon boundary.
- GPR_$CLOSE_FILL_PGON -- closes and fills the currently open polygon.
- GPR_$CLOSE_RETURN_PGON -- closes the currently open polygon and returns the list of trapezoids within its interior.

5.3 TEXT OPERATIONS

Using the graphics package, a program can mix text characters and graphic images in a single bitmap in a Display Manager frame, an acquired window, the borrowed display, or main memory. The text routines are:

- GPR_$LOAD_FONT_FILE -- loads a font from a file into the font storage area of display memory. A single program may load multiple fonts and then set them for use, one at a time.
- GPR_$UNLOAD_FONT_FILE -- unloads a font.
- GPR_$SET_CHARACTER_WIDTH -- set the parameter WIDTH of the specified character in the specified font.
- GPR_$INQ_CHARACTER_WIDTH -- returns the width of the specified character in the specified font.
- GPR_$SET_HORIZONTAL_SPACING -- sets the parameter for the width of spacing between displayed characters for the specified font.
- GPR_$INQ_HORIZONTAL_SPACING -- returns the parameter for the width of spacing between displayed characters for the specified font.
- GPR_$INQ_SPACE_SIZE -- returns the width of the space to be displayed when a character requested is not in the specified font.
- GPR_$REPLICATE_FONT -- Creates and loads a modifiable copy of a font.
- GPR_$SET_SPACE_SIZE -- specifies the width of the space to be displayed when a character requested is not in the specified font.
- GPR_$SET_TEXT_FONT -- selects a loaded font for use in subsequent text operations.
- GPR_$INQ_TEXT -- returns the descriptor of the currently set text font.
- GPR_$SET_TEXT_PATH -- specifies the direction in which a line of text is written.
- GPR_$INQ_TEXT_PATH -- returns the direction for writing a line of text.
- GPR_$SET_TEXT_VALUE -- specifies the pixel value to use for writing text.
- GPR_$SET_TEXT_BACKGROUND_VALUE -- specifies the pixel value to use for text background.

- GPR_$INQ_TEXT_VALUES -- returns the text and text background pixel values.

- GPR_$TEXT -- writes text in the current bitmap, beginning at the current position and proceeding in the direction specified by the most recent use of GPR_$SET_TEXT_PATH.

- GPR_$INQ_TEXT_EXTENT -- returns the width and height, in pixels, of the area a text string would span if it were written with GPR_$TEXT.

- GPR_$INQ_TEXT_OFFSET -- returns the x- and y-offsets from the top left pixel of a string to be written by GPR_$TEXT to the origin of its first character. This routine also returns the x- or y-offset to the pixel which is the new current position after the GPR_$TEXT call. This is the y-offset when the text path is vertical.

CHAPTER 6

GRAPHIC BLOCK TRANSFERS: BLTs

This chapter describes three types of bit block transfers (BLTs) and their uses. The differences between calls to the graphics primitives BLTs and display driver BLTs are also presented.

6.1 FUNCTION OF BLTs

A program can move a rectangular section (a window) of a bitmap from one location in main memory to another, one location in display memory to another, or back and forth between locations in display and main memory. The display hardware and graphics primitives software implement these moves with bit block transfers (BLTs).

A program can use the BLT, for example, if it has manipulated bitmaps in main memory and needs to copy them to display memory. Or a program might establish a bitmap with a halftone pattern and then use a BLT to combine the halftone bitmap with an image drawn on another bitmap, to give the image a shading effect.

6.1.1 Bit BLTs, Pixel BLTs, and Additive BLTs

The three types of graphics primitives BLT operations are as follows.

- A bit BLT is an operation that moves a single plane of the source bitmap to a single plane of the destination bitmap.
- A pixel BLT is an operation that moves all planes of the source bitmap to the corresponding planes of the destination bitmap.
- An additive BLT is an operation that moves a single plane of the source bitmap to all unmasked planes of the destination bitmap. A program can, for example, store character fonts and graphic templates in a compact, monochromatic form in a single plane of a bitmap. Later, the program can select various colors for various instances of the template by performing additive BLTs on the single-plane template bitmap and the appropriately colored multi-plane bitmap.

6.1.2 Using a Plane Mask With a BLT

A program can mask planes of a bitmap to establish the following:

- source planes of a pixel BLT operation
- destination planes of a pixel BLT operation
- destination planes of an additive BLT operation

For plane masking procedures, see the routine GPR_$SET_PLANE_MASK.

6.1.3 Using Raster Operations With a BLT

When a program invokes a BLT with the default raster operation, the BLT moves the rectangle and retains all bit values. When the program uses a BLT with any other raster operation, the BLT combines two rectangles and assigns the resultant bit values according to the raster operation of the destination bitmap.

6.1.4 BLTs for Graphics Primitives and for the Display Driver

Table 6-1 shows the differences between calls to the BLTs for graphics primitives and for the display driver. For additional information about BLTs for the display driver see the DOMAIN System Programmer's Reference.

Table 6-1. Characteristics of Bit Block Transfers

Calls to BLTs	
For Graphics Primitives	For the Display Driver
Source and destination bitmaps can be rectangles of different sizes.	Source and destination bitmaps must be conforming rectangles.
Starting coordinate position (origin) and height and width define a source rectangle (window). Source and destination rectangles can have different origins.*	Start and end positions define both the source and destination arrays.
The origin of a bitmap is always at the top left corner.	The origin of the bitmap can be any corner.
BLTs can move rectangles between display and main memory.	BLTs can move arrays only from one area of display memory to another.
*See Figure 6-1.	

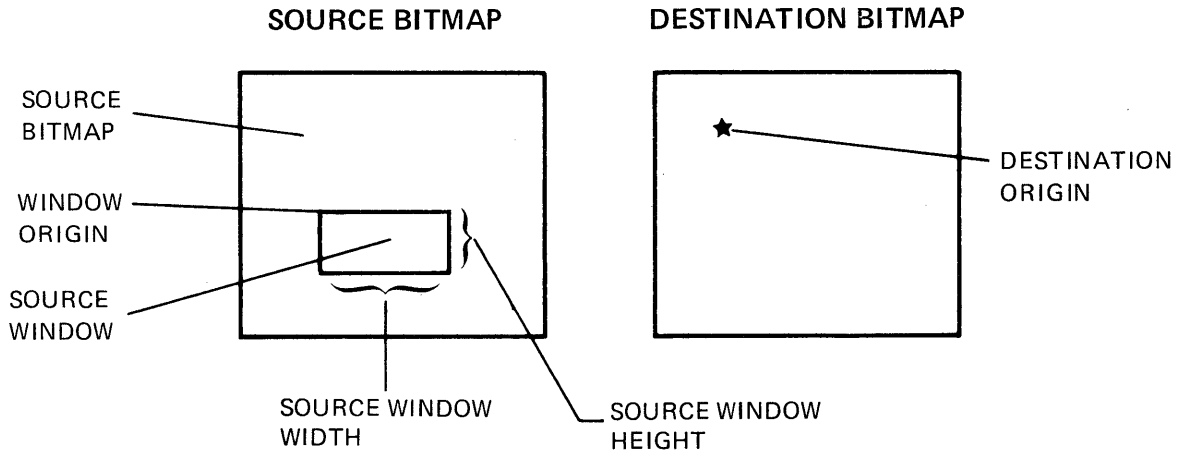


Figure 6-1. Information Required for Graphics BLT

6.2 THE BLT ROUTINES

The graphics BLT routines are as follows:

- GPR_$PIXEL_BLT -- performs pixel block transfers from any bitmap to the current bitmap.
- GPR_$BIT_BLT -- performs a bit block transfer from a single plane of any bitmap to a single plane of the current bitmap.
- GPR_$ADDITIVE_BLT -- adds a single plane of any bitmap to all unmasked planes of the current bitmap.

6.2.1 Example of a BLT Operation

In a BLT operation, bits are transferred only on the rectangular area in which the source bitmap, source window, and destination clipping window intersect (see Figure 6-2). Nothing is transferred outside the bounds of the bitmap. For example, if the clipping window of the current bitmap (the destination bitmap) excludes part of the destination rectangle that would otherwise receive pixels, the size of the actual rectangle moved will be smaller than the source window. Similarly, if the source window overflows the boundaries of the source bitmap, the size of the actual rectangle moved will be smaller than the source window.

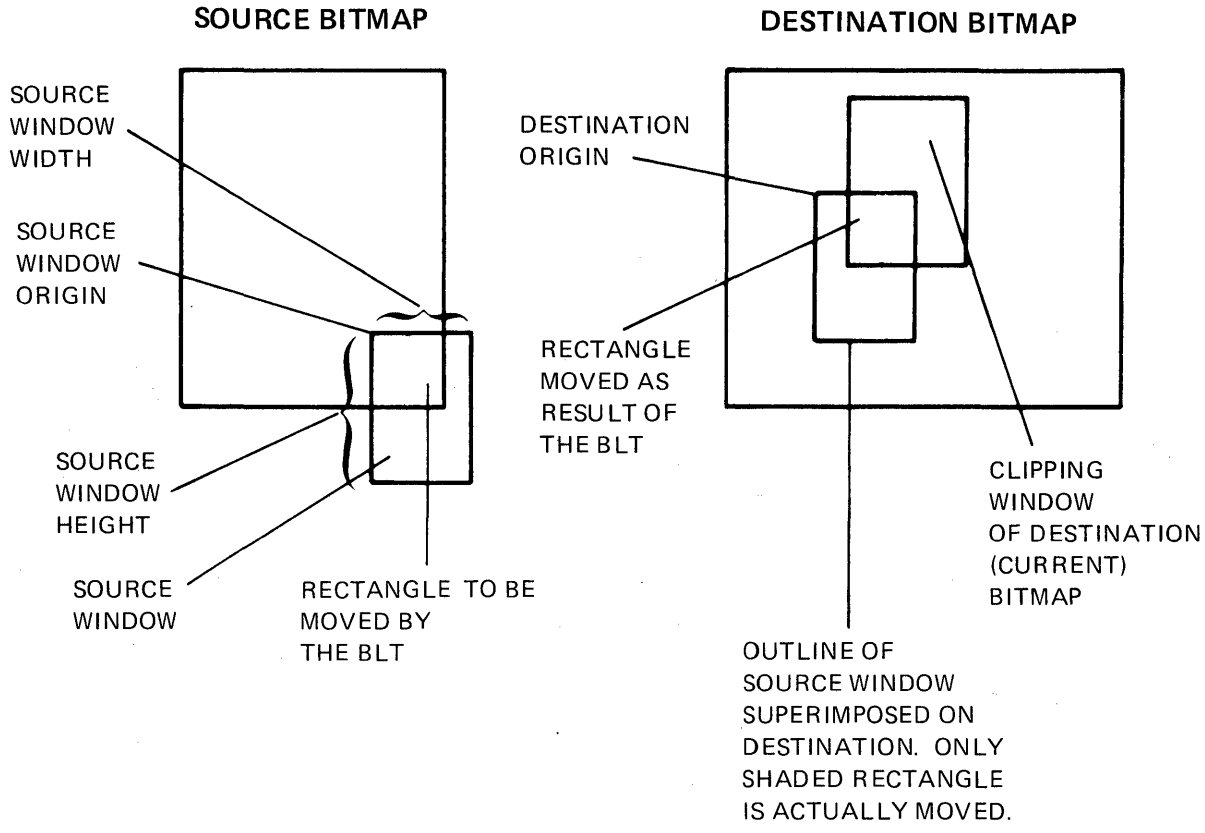


Figure 6-2. BLT Example: Intersection of Source Bitmap, Source Window, Destination Clipping Window

6.2.2 Example of a BLT With a Raster Operation

Figure 6-3 shows a source bitmap in main memory, a destination bitmap in display memory, and the bitmap created by using a BLT with raster operation 1, the logical "and" function. The figure shows 0 bits as black, and 1 bits as white.

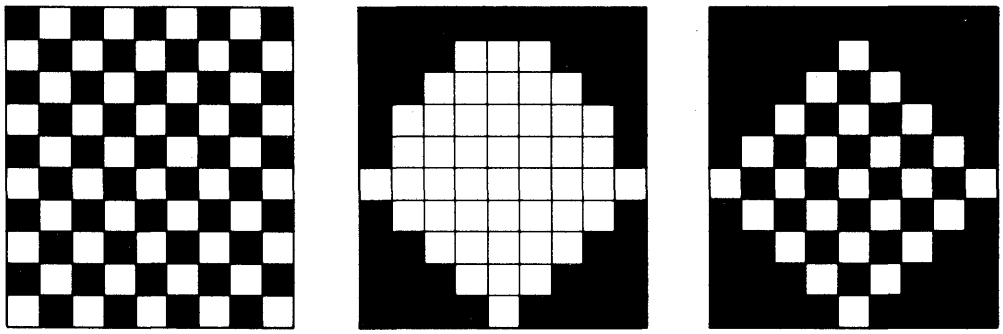


Figure 6-3. Example of BLT With Raster Op Code = 1 (Logical "and")

CHAPTER 7

CURSOR CONTROL AND INPUT OPERATIONS

This chapter describes cursor control and input operations. The input routines synchronize program execution around input events. These events include keystroke, mouse or puck buttons, locator and locator stop from mouse or touchpad, and window transition.

7.1 USING CURSOR CONTROL

The complete set of cursor routines is available in borrow-display and direct mode. In frame mode, the cursor is controlled by the Display Manager and is always displayed. Therefore, in frame mode, you can change only the cursor's position. Cursor routines include the following:

- GPR_$SET_CURSOR_ACTIVE -- specifies whether to display the cursor. Initially, the cursor is disabled.
- GPR_$SET_CURSOR_PATTERN -- sets a bitmap pattern as the cursor pattern. This bitmap can be a maximum of 16x16 pixels. The initial cursor size varies, depending on the standard font the Display Manager uses.
- GPR_$SET_CURSOR_POSITION -- sets a position on the screen for display of the cursor. The initial cursor position is (0,0). Programs running in frame mode can call this routine.
- GPR_$SET_CURSOR_ORIGIN -- designates one of the cursor's pixels as the cursor origin. Thereafter, when the cursor is moved, the pixel designated as the cursor origin moves to the screen coordinate designated as the cursor position, as shown in Figure 7-1.

7.1.1 Implementation Restrictions on the Cursor

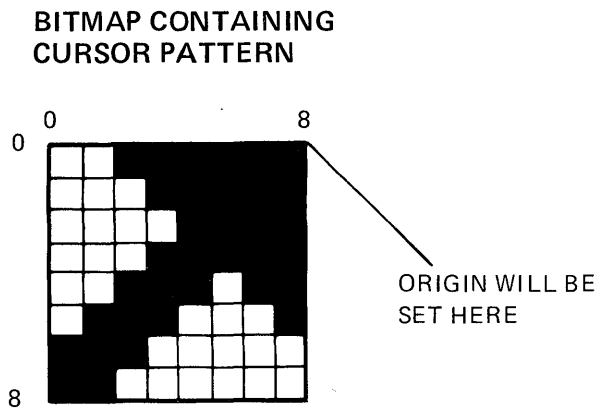
When the cursor is active, the cursor pattern is stored in display memory. Therefore, programs that operate in borrow-display or direct mode, have the potential to interfere with the cursor pattern and/or to cause the cursor to interfere with a bitmap pattern. To avoid this problem, disable the cursor

before performing output procedures to any area of the display in which the cursor could be located.

7.1.2 Display Mode and Cursor Control

In borrow-display and direct mode, the program has complete control over the cursor. In direct mode, the program-defined cursor pattern and origin are in effect only within the direct mode window. As the keyboard user moves between the direct window and other windows on the screen, the system automatically changes the cursor pattern.

If the program executes in frame mode, program control of the cursor is limited. The only cursor control routine that operates in frame mode is GPR_$SET_CURSOR_POSITION, and the program can move the cursor with this routine only if it lies within the frame when GPR_$SET_CURSOR_POSITION is called.




```
.
.
.
CALL GPR_$SET_CURSOR_ORIGIN (8,0,STATUS)
CALL GPR_$SET_CURSOR_POSITION (400,400, STATUS)
.
.
.
```

Figure 7-1. Cursor Origin Example

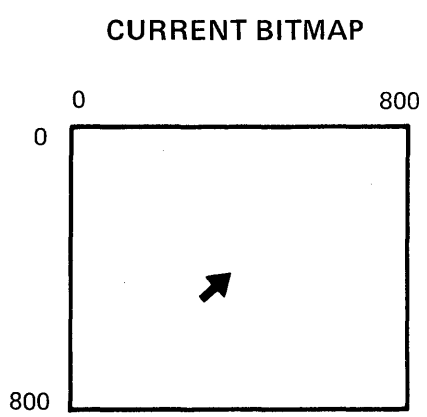


Figure 7-1. Cursor Origin Example (continued)

7.2 USING INPUT OPERATIONS

The graphics primitives package includes a set of routines that enable graphics programs to accept input from various input devices. The input routines synchronize program execution around input events. Input routines function in all display modes except GPR_$NO_DISPLAY.

7.2.1 Event Types

An event occurs when input is generated in a frame, window, or borrowed display. The GPR package supports several classes of event, called event types. Programs use an input routine to select the type of event to be reported to them; this operation is called enabling an event type. The event types are the following:

- Keystroke event — A keystroke event occurs when you type specified keyboard characters. Programs can select a subset of keyboard characters, called a keyset, to be recognized as keystroke events. Except in borrow-display mode, keys that do not belong to the keyset are processed normally by the Display Manager.

- Button event — A button event occurs when you press a button on the mouse or bitpad puck.

- Locator event — A locator event occurs when you move the mouse or use the touchpad or bitpad.

- Locator stop event — A locator stop event occurs when you stop moving the mouse or stop using the touchpad or bitpad.

- Window transition event — Except in borrow-display mode, the cursor

may move into and out of the window in which GPR input is being performed. When the cursor leaves a window used for graphics display, the input routines report to the program an event of type GPR_$LEFT_WINDOW. When the cursor enters the window, the routines report an event type of GPR_$ENTERED_WINDOW.

Enabled input events are stored in attribute blocks (not with bitmaps) in much the same way as attributes are. However, you cannot set and inquire about input events in the same way that you can attributes. You use GPR_$ENABLE_INPUT and GPR_$DISABLE_INPUT instead of GPR_$SET... and GPR_$INQ.... The effect of this difference is the following. When a program changes attribute blocks for a bitmap during a graphics session, the input events you enabled are lost unless you enable those events for the new attribute block.

7.2.2 Event Reporting

If an event type is enabled, the input routines will report each event of the enabled type to the program with a cursor position. This position is relative to the upper left corner of the window.

If the enabled event type is keystroke or button, the input routines will return an ASCII character from the enabled keyset. When defining a keyset for a keystroke event, consult the system insert files /SYS/INS/KBD.INS.PAS, /SYS/INS/KBD.INS.FTN and /SYS/INS/KBD.INS.C. These files contain the definitions for the non-ASCII keyboard keys in the range 128 through 255.

The input routines report mouse and bitmap button events as ASCII characters. "Down" transitions range from "a" to "d"; "up" transitions range from "A" to "D". The three mouse keys start with (a/A) on the left side. As with keystroke events, button events can be selectively enabled by specifying a button keyset.

Locator events merely report the x- and y-coordinates of the locator input. If the program has not enabled locator events, the GPR software handles any locator data itself by moving the arrow cursor around the window. At the next occurrence of an enabled event, the GPR software reports the locator final cursor position to the program as well as the enabled event.

As noted above, enabled input events are stored in attribute blocks (not with bitmaps) in much the same way as attributes are. When a program allocates more than one attribute block, different sets of events are associated with each attribute block. The events enabled for a particular bitmap are the events stored in the attribute block for that bitmap. You must enable the desired events for each window.

GPR_$ENABLE_INPUT and GPR_$DISABLE_INPUT work on the attribute block of the following bitmap: the current bitmap if it is a screen bitmap; otherwise, the screen bitmap which was most recently current.

When you have more than one bitmap displayed, you can determine the source of input in either of two ways:

-Make certain that you have enabled entered-window events for all windows. Then remember which window was the last entered. This window is the source of the input event.

-Use GPR_$INQ_WINDOW_ID after each input event.

You can assign a distinct character to identify each window. To do this, use GPR_$SET_WINDOW_ID. This allows you to determine which window was entered when an input event occurs of the entered-window type.

7.2.3 Input Routines

The graphics primitives provide the following routines to perform input operations:

- GPR_$ENABLE_INPUT -- enables events of a specific event type. If the event type is keystroke or button, the routine also enables a specific keyset to select which keys or buttons will generate input events. Programs must call this routine once for each event type to be enabled.

- GPR_$DISABLE_INPUT -- disables events for the event type previously enabled with GPR_$ENABLE_INPUT.

- GPR_$EVENT_WAIT -- suspends program execution until one of the events enabled by GPR_$ENABLE_INPUT occurs. If the event type is keystroke or button, this routine waits until a member of the specified keyset is input. The information returned includes the type of event that occurred, the character (if any) associated with the event, and the position at which the event occurred. The position will be relative to the upper left corner of the window, or, if the mode is borrow-display, the screen.

- GPR_$COND_EVENT_WAIT -- Performs the same function as GPR_$EVENT_WAIT except that if no event has occurred, the routine returns to the program immediately with an event type which indicates that no event has occurred (GPR_$NO_EVENT).

- GPR_$GET_EC -- returns the eventcount associated with a graphic input event. Programs can use this routine in conjunction with GPR_$COND_EVENT_WAIT to wait for a combination of system events as well as GPR input events.

- GPR_$SET_INPUT_SID -- establishes a selected stream as the standard input stream. The default standard input stream is STREAM_$STDIN. Programs can only use this call in frame mode. In borrow-display and direct modes, input comes directly from the keyboard.

- GPR_$SET_WINDOW_ID — establishes the character that identifies the current bitmap's window. This character is returned by GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT when they return GPR_$ENTERED_WINDOW events. The character indicates which window was entered.

- GPR_$INQ_WINDOW_ID — returns the character that identifies the current bitmap's window.

The calling sequence for each of the input routines is described with the other GPR routines in Chapter 11.

CHAPTER 8

DIRECT GRAPHICS IN WINDOWS

This chapter describes the routines that graphics programs can call to perform input and output to windows. A sample program is included to show the input and output calls with other GPR routines.

8.1 USING DIRECT MODE

Programs that select direct mode at initialization (with GPR_$INIT) are allowed exclusive access to Display Manager windows for graphics operations. In this mode, the GPR software automatically translates and clips to window boundaries all graphics operations, except program access to display memory. The window boundary used by the GPR software is the inner boundary of the Display Manager window, which is five pixels away from the window border seen on the screen.

The next sections describe the routines that graphics programs can call to perform input and output to windows. The calling syntax for each routine is described with the other GPR routines in Chapter 11.

8.1.1 Direct Graphics Output

To perform graphics output to a direct window, programs must acquire exclusive access to display operations within the window by calling the routine GPR_$ACQUIRE_DISPLAY. Programs can only acquire the display for a limited amount of time; the default time limit is one minute. However, programs can modify the time limit with the routine GPR_$SET_ACQ_TIME_OUT. The maximum amount of time that a display can be acquired is five minutes.

In direct mode, the program must acquire the display before calling the following routines:

- GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT
- GPR_$LINE, GPR_$POLYLINE, GPR_$MULTILINE
- GPR_$TEXT

- GPR_$PIXEL_BLT, GPR_$ADDITIVE_BLT, and GPR_$BIT_BLT
- GPR_$CLEAR, GPR_$READ_PIXELS, and GPR_$WRITE_PIXELS
- GPR_$RECTANGLE, GPR_$TRIANGLE, GPR_$TRAPEZOID, GPR_$MULTITRAPEZOID, GPR_$CIRCLE, GPR_$CIRCLE_FILLED, GPR_$DRAW_BOX, GPR_$ARC_3P, GPR_$SPLINE_CUBIC_P, GPR_$SPLINE_CUBIC_X, and GPR_$SPLINE_CUBIC_Y.
- GPR_$START_PGON, GPR_$PGON_POLYLINE, GPR_$CLOSE_FILL_PGON and GPR_$CLOSE_RETURN_PGON
- GPR_$ALLOC_HDM_BITMAP
- GPR_$ENABLE_DIRECT_ACCESS, GPR_$WAIT_FRAME, GPR_$SET_COLOR_MAP and GPR_$REMAP_COLOR_MEMORY

The program can call the other graphics primitives routines at any time in direct mode, including the cursor routines and any of the set and inquire routines. In addition, if the bitmap(s) involved in the drawing calls are memory bitmaps (as opposed to screen bitmaps), the display need not be acquired.

GPR_$ACQUIRE_DISPLAY establishes exclusive access to display operations within a window. If the display is already acquired when this call is made, a count of calls is incremented so that pairs of acquire/release calls can be nested. A graphics program releases an acquired display by calling either GPR_$RELEASE_DISPLAY or GPR_$FORCE_RELEASE. The first routine releases the display only when the value of the internal counter is equal to one; otherwise, it decrements the counter.

If more than one program module has acquired the display, calling GPR_$RELEASE_DISPLAY guarantees that the display will not actually be released until all the modules have individually "released" it. The routine GPR_$FORCE_RELEASE releases the display regardless of whether or not other program modules are currently using it.

In general, a program creates a graphic image in direct mode by acquiring a display, performing some output, and releasing the display before the timeout expires. The program repeats this sequence until the session is completed.

If the program has not released the display when the time-out expires and another process (for example, the Display Manager) needs the display, an acquire time-out fault (SMD_$ACQUIRE_TIMEOUT) is generated in the user process. The acquire time-out fault is a warning fault that the program can intercept with a clean-up handler or static fault handler. If the program does not release the display within a few seconds of the acquire time-out fault, a second fault occurs (with the status code FAULT_$QUIT) and the program loses control of the display.

8.1.2 Direct Graphics Input

Programs can also call GPR input routines in direct mode, as well as in frame and in borrow-display modes. When performing input in direct mode, the program does not need to acquire and release the display explicitly with GPR_$ACQUIRE_DISPLAY and GPR_$RELEASE_DISPLAY. Instead, the program makes one call to GPR_$ACQUIRE_DISPLAY. Then, when the program calls the routine GPR_$EVENT_WAIT or GPR_$COND_EVENT_WAIT, the routine automatically releases and reacquires the display.

8.1.3 Keyboard Acquisition and Release

When a program acquires a display, it automatically acquires control of the keyboard from the Display Manager; releasing the display automatically returns keyboard control to the Display Manager.

If the program controls both the keyboard and the display, typing CTRL/Q will cause a process fault with status code FAULT_$QUIT. On the first CTRL/Q, no further action is taken. If a second CTRL/Q is typed with no intervening input, the keyboard is forcibly returned to the Display Manager, a second fault occurs, and the program loses the display.

This CTRL/Q action occurs regardless of whether or not CTRL/Q is included in the keyset specified in the call to GPR_$ENABLE_INPUT. To disable the CTRL/Q function, the program must call the display driver interface routine SMD_$SET_QUIT_CHAR.

8.1.4 Obscured Windows

As a result of overall screen activity, such as window push/pop, move/grow, and window creation, the window associated with the display to be acquired can sometimes be obscured. The routine GPR_$SET_OBSCURED_OPT allows the program to specify the action to be taken when the acquire routine encounters an obscured window:

- Pop the window and acquire the display
- Suspend display acquisition until the window becomes unobscured
- Return to the program without acquiring the display
- Acquire the display even if the window is obscured

If the program chooses to acquire an obscured display, it can use the routine GPR_$INQ_VIS_LIST to locate any unobscured parts of the window. The GPR

software does not automatically clip drawing operations to the visible parts of an obscured window. The program must perform this operation with the routine GPR_$SET_CLIP_WINDOW.

8.1.5 Image Redisplay

In direct mode, the use of keyboard actions such as push/pop or CTRL/F can require that a program redraw the image displayed in a direct-mode window. Graphics programs can refresh an image using one of two methods:

- The program can request that the Display Manager save the bitmap image in the window and restore it when necessary with the routine GPR_$SET_AUTO_REFRESH. This method reduces performance but is the easier of the two methods to use.

- The program can provide its own procedure to redraw the image based on data the program has saved. The program calls GPR_$SET_REFRESH_ENTRY to specify the entry point for the procedure. The procedure is called by the GPR software whenever the display is reacquired and the Display Manager has indicated that the refresh is needed. The display is either reacquired explicitly by GPR_$ACQUIRE_DISPLAY or implicitly by GPR_$EVENT_WAIT.

A call to GPR_$SET_AUTO_REFRESH or GPR_$SET_REFRESH_ENTRY applies to the current bitmap. However, the data from these calls is stored in attribute blocks (not with bitmaps) in much the same way as attributes are stored. This means that when a program changes attribute blocks for a bitmap during a graphics session, the auto refresh and refresh entry procedures are lost unless you set them for the new attribute block.

8.1.6 Program Access to Display Memory

As in borrow-display mode, programs may access display memory directly, using the call GPR_$INQ_BITMAP_POINTER. Because the window may not always be aligned on a word boundary, the program must call the routine GPR_$INQ_BM_BIT_OFFSET to establish the window offset from the nearest word boundary. The GPR software cannot clip program access to display memory; the program must perform this operation itself using the data returned by GPR_$INQ_BITMAP_DIMENSIONS.

8.1.7 Hidden Display Memory

To get access to hidden display memory when you are using direct mode, you must acquire the display and use the routine GPR_$ALLOCATE_HDM_BITMAP. A program running in an acquired display has exclusive access to hidden display memory. However, when the program releases the window, the contents of

hidden display memory are no longer protected from modification by other programs. The contents of hidden memory remain intact only so long as no other program intervenes.

A graphics program can provide its own procedure to reconstruct hidden display memory; the routine GPR_$SET_REFRESH_ENTRY allows the program to specify the entry point of the procedure. When the program reacquires the display (either explicitly with GPR_$ACQUIRE_DISPLAY or implicitly with GPR_$EVENT_WAIT), GPR software calls the user-written procedure to reload hidden display memory, if another program or the Display Manager has used it in the meantime.

If the graphics program use GPR_$SET_REFRESH_ENTRY to specify both a window refresh procedure and a hidden display memory refresh procedure, the hidden display refresh procedure is called first.

Any fonts loaded by a program during execution in an acquired display are automatically reloaded when the program reacquires the display.

8.2 PROGRAM EXAMPLE

This section contains a Pascal program that uses direct mode to handle input from a user. The program expects the user to do the following:

- Wait until the display is acquired, at which time a cursor appears
- Type in six characters
- Operate the touchpad

At any time, typing the character 'q' exits from the program.

```
PROGRAM DIRECT_EXAMPLE;

%include '/SYS/INS/BASE.INS.PAS';
%include '/SYS/INS/GPR.INS.PAS';


VAR
    event: gpr_$event_t;
    sts: status_$t;
    cur_position: gpr_$position_t;
    event_type: gpr_$event_t;
    ch: char;
    i: integer;
    timeout: time_$clock_t;
    disp_bm_size: gpr_$offset_t;
    init_bitmap: gpr_$bitmap_desc_t;
    unobscured: Boolean;
    fwidth, fhite: integer;
    fname: pad_$string_t;
    fnsize: integer;
    fnlen: integer;
    fid: integer;
    start: gpr_$offset_t;
    xend: integer;

BEGIN

    { initialize specifying direct mode }

    disp_bm_size.x_size := 1024;
    disp_bm_size.y_size := 1024;
    gpr_$init( gpr_$direct, stream_$stdout, disp_bm_size, 0, init_bitmap, sts );


    { set up text font that will be used in direct window }

    pad_$inq_font( stream_$stdout, fwidth, fhite, fname, 80, fnlen, sts );
    gpr_$load_font_file( fname, fnlen, fid, sts );
    gpr_$set_text_font( fid, sts );


    { set time-out to 5 seconds }

    timeout.low32 := 5*250000;
    timeout.high16 := 0;
    gpr_$set_acq_time_out(timeout, sts);


    { set obscured option to pop the window if it is currently obscured }

    gpr_$set_obscured_opt( gpr_$pop_if_obs, sts );
```

```
{ turn on auto refresh so that the DM will refresh the screen when needed }

gpr_$set_auto_refresh( true, sts );

{ acquire the display.  If the window is obscured, it will be popped because
  of the setting of the obscured option }

unobscured := gpr_$acquire_display( sts );


{ enable keystroke event and characters from 0 to 127 which includes all
  white keys }

gpr_$enable_input( gpr_$keystroke, [chr(0)..chr(127)], sts );
gpr_$set_cursor_active( true, sts );


FOR i := 1 TO 6 DO
BEGIN

    { call event wait and wait for a keystroke event, char, and cursor pos }

    unobscured := gpr_$event_wait( event, ch, cur_position, sts );


    { print char at present cursor position and then move the cursor
      to the next position }

    IF event = gpr_$keystroke THEN
    BEGIN
        IF ch = 'q' THEN RETURN;
        gpr_$set_cursor_active( false, sts );
        gpr_$move( cur_position.x_coord, cur_position.y_coord, sts );
        gpr_$text( ch, 1, sts );

        { determine width of character from font, and move
          the cursor by that amount in preparation for the
          next input character }

        gpr_$inq_text_offset( ch, 1, start, xend, sts );
        cur_position.x_coord := cur_position.x_coord + xend;
        gpr_$set_cursor_position( cur_position, sts );

        gpr_$set_cursor_active( true, sts );
   END;

END;

{ disable keystroke event }

gpr_$disable_input( gpr_$keystroke, sts );
```

```
    { enable keystroke event and the letter 'q' }

    gpr_$enable_input( gpr_$keystroke, ['q'], sts );
    gpr_$set_cursor_active( true, sts );


    { enable locator events }

    gpr_$enable_input( gpr_$locator, [], sts );


    { call event wait and wait for locator events. Follow cursor returned
      from event wait using gpr_$set_cursor_position. To quit, type 'q' }

    WHILE TRUE DO
    BEGIN
        unobscured := gpr_$event_wait( event, ch, cur_position, sts );
        IF event = gpr_$locator THEN
            gpr_$set_cursor_position( cur_position, sts )
        ELSE IF event = gpr_$keystroke THEN EXIT;
    END;


    { disable events }

    gpr_$disable_input( gpr_$keystroke, sts );
    gpr_$disable_input( gpr_$locator, sts );


    { terminate direct mode graphics }

    gpr_$terminate( false, sts );


END.
```

CHAPTER 9

PROGRAMMING INFORMATION

This chapter lists insert files, data structures with their argument type, and error messages.

9.1 INSERT FILES

Programs must include the following file, appropriate to their programming language, to use system routines:

FORTRAN	Pascal	C
/SYS/INS/BASE.INS.FTN	/SYS/INS/BASE.INS.PAS	/SYS/INS/BASE.INS.C

To use graphics primitives routines, programs must also include the following file, appropriate to their programming language. These files contain constant definitions required by the routines.

FORTRAN	Pascal	C
/SYS/INS/GPR.INS.FTN	/SYS/INS/GPR.INS.PAS	/SYS/INS/GPR.INS.C

9.2 DATA STRUCTURES

The graphics primitives use many common data structures. These are listed with their data types. The data structures and data types are described more completely in Chapter 11 with the routines in which they occur.

9.2.1 Data Structures: PASCAL and C

The data structures and data types shown below are defined in the insert files for Pascal and C.

Data Structure	Data Type
GPR_$ACCESS_MODE_T	one of: GPR_$CREATE, GPR_$UPDATE, GPR_$WRITE, GPR_$READONLY
GPR_$ATTRIBUTE_DESC_T	4-byte integer
GPR_$BMF_GROUP_HEADER_ARRAY_T	array [0..GPR_$MAX_BMF_GROUP] of GPR_$BMF_GROUP_HEADER_T
GPR_$BMF_GROUP_HEADER_T	record of six elements: N_SECTS 2-byte integer, 1..8 PIXEL_SIZE 2-byte integer ALLOCATED_SIZE 2-byte integer BYTES_PER_LINE 2-byte integer BYTES_PER_SECT 4-byte integer STORAGE_OFFSET 4-byte integer
GPR_$BITMAP_DESC_T	4-byte integer
GPR_$COLOR_T	4-byte integer
GPR_$COLOR_VECTOR_T	array of GPR_$COLOR_T
GPR_$COORDINATE_ARRAY_T	array of GPR_$COORDINATE_T
GPR_$COORDINATE_T	2-byte positive integer
GPR_$DIRECTION_T	one of: GPR_$UP, GPR_$DOWN, GPR_$LEFT, GPR_$RIGHT
GPR_$DISPLAY_CONFIG_T	one of: GPR_$BW_800x1024, GPR_$BW_1024x800, GPR_$COLOR_1024X1024x4, GPR_$COLOR_1024X1024x8
GPR_$DISPLAY_MODE_T	one of: GPR_$BORROW, GPR_$BORROW_NC GPR_$FRAME, GPR_$NO_DISPLAY, GPR_$DIRECT
GPR_$EC_KEY_T	GPR_$INPUT_EC

GPR_$EVENT_T	one of: GPR_$KEYSTROKE, GPR_$BUTTONS, GPR_$LOCATOR, GPR_$ENTERED_WINDOW, GPR_$LEFT_WINDOW, GPR_$LOCATOR_STOP, GPR_$NO_EVENT
GPR_$HORIZ_SEG_T	record of three elements: .(x_coord_l, x_coord_r, y_coord): GPR_$COORDINATE_T
GPR_$KEYSET_T	set of characters
GPR_$IMAGING_FORMAT_T	one of: GPR_$INTERACTIVE, GPR_$IMAGING_1024x1024x8, GPR_$IMAGING_512x512x24
GPR_$LINE_PATTERN_T	four element array of 2-byte integers
GPR_$LINESTYLE_T	one of: GPR_$SOLID, GPR_$DOTTED
GPR_$MASK_T	set of 0..7
GPR_$OBSCURED_OPT_T	one of: GPR_$OK_IF_OBS, GPR_$ERR_IF_OBS, GPR_$POP_IF_OBS, GPR_$BLOCK_IF_OBS
GPR_$OFFSET_T	record of two elements: .(x_size, y_size): 0..32767
GPR_$PIXEL_ARRAY_T	array of 4-byte integers
GPR_$PIXEL_VALUE_T	4-byte integer
GPR_$PLANE_T	2-byte integer
GPR_$POSITION_T	record of two elements: .(x_coord, y_coord): GPR_$COORDINATE_T
GPR_$RASTER_OP_ARRAY_T	array [0..7] of GPR_$RASTER_OP_T
GPR_$RASTER_OP_T	0..15

GPR_$STRING_T	array of characters
GPR_$TRAP_T	record of two elements: .(top, bot): GPR_$HORIZ_SEG_T .(x_coord_l, x_coord_r, y-coord): GPR_$COORDINATE_T
GPR_$TRAP_LIST_T	array of GPR_$TRAP_T
GPR_$VERSION_T	record of two elements: .(major_version, minor_verson): integer;
GPR_$WINDOW_T	record of two elements: .window_base: GPR_$POSITION_T .(x_coord, y_coord): GPR_$COORDINATE_T .window_size: GPR_$OFFSET_T .(x_size, y_size): 0..32767
GPR_$WINDOW_LIST_T	array of GPR_$WINDOW_T

9.2.2 Data Structures: FORTRAN

The following information on data structures and data types is for use by FORTRAN programmers. Constants corresponding to the Pascal enumerated types are declared in the FORTRAN insert file.

GPR_$ACCESS_MODE_T describes the mode of access to external bitmap objects as one of the following:

GPR_$CREATE,
GPR_$UPDATE,
GPR_$WRITE,
GPR_$READONLY.

This is an I*2 variable.

GPR_$ATTRIBUTE_DESC_T describes the unique descriptor of an attribute block. This is an I*2 variable.

GPR_$BMF_GROUP_HEADER_ARRAY_T describes the array [0..GPR_$MAX_BMF_GROUP] of GPR_$BMF_GROUP_HEADER_T for group headers of an external bitmap.

GPR_$BMF_GROUP_HEADER_T describes the group headers of the external bitmap as a record of six elements. In FORTRAN, use the following equivalences:

Parameter	(n_groups =1)
I*2	headers2(8,n_groups)
I*4	headers4(4,n_groups)
Equivalence	headers2(1,1),headers4(1,1)
N_SECTS	headers2(1,2)
PIXEL_SIZE	headers2(2,n)
ALLOCATED_SIZE	headers2(3,n)
BYTES_PER-LINE	headers2(4,n)
BYTES_PER-SECT	headers4(3,n)
STORAGE_OFFSET	headers4(4,n)

GPR_$BITMAP_DESC_T describes the unique descriptor of a bitmap. This is an I*2 variable.

GPR_$COLOR_T describes a color value. This is an I*2 variable.

GPR_$COLOR_VECTOR_T describes a list of color values. This list contains 4-byte integers, each designating a color value. Use a declaration like the following:

```
      I*4 COLOR_VECTOR (n)
C     (1) = FIRST_COLOR_VALUE
C     (2) = SECOND_COLOR_VALUE
            .
            .
            .
```

GPR_$COORDINATE_ARRAY_T describes a list of either x-coordinates or y-coordinates. It contains 2-byte integers. The following FORTRAN example declares a variable of this type.

```
      I*2 COORDINATE_ARRAY (n)
C     (1) = FIRST_X_COORD
C     (2) = SECOND_X_COORD
            .
            .
            .
```

GPR_$DIRECTION_T describes the direction of movement from one text character position to the next in the current bitmap as one of the following:

```
GPR_$UP,
GPR_$DOWN,
GPR_$LEFT,
GPR_$RIGHT
```

This is an I*2 variable.

GPR_$DISPLAY_CONFIG_T describes the display configuration as one of the following:

```
GPR_$BW_800x1024,
GPR_$BW_1024x800,
GPR_$COLOR_1024X1024x4,
GPR_$COLOR_1024X1024x8
```

This is an I*2 variable.

GPR_$DISPLAY_MODE_T describes the display mode as one of the following:

```
GPR_$BORROW,
GPR_$BORROW_NC,
GPR_$FRAME,
GPR_$NO_DISPLAY,
GPR_$DIRECT
```

This is an I*2 variable.

GPR_$EC_$KEY_T specifies which event count to obtain as GPR_$INPUT_EC. This is an I*2 variable.

GPR_$EVENT_T specifies input event types as one of the following:

```
GPR_$KEYSTROKE,
GPR_$BUTTONS,
GPR_$LOCATOR,
GPR_$ENTERED_WINDOW,
GPR_$LEFT_WINDOW,
GPR_$LOCATOR_STOP,
GPR_$NO_EVENT
```

This is an I*2 variable.

GPR_$HORIZ_SEG_T describes a horizontal line segment on the screen. It contains three 2-byte integers. The first designates the left x-coordinate, the second corresponds to the right x-coordinate, and the third designates the y-coordinate. In FORTRAN, use a declaration like the following:

```
      I*2 HORIZONTAL_SEGMENT (3)
C     (1) = LEFT_X_COORD
C     (2) = RIGHT_X_COORD
C     (3) = Y_COORD
```

GPR_$IMAGING_FORMAT_T specifies pixel and bit values with one of the following:

```
GPR_$INTERACTIVE,
GPR_$IMAGING_1024x1024x8,
GPR_$IMAGING_512x512x24
```

This is an I*2 variable.

GPR_$KEYSET_T describes a set of characters associated with the graphics input event types GPR_$KEYSTROKE and GPR_$BUTTONS. For a procedure to use in building a set of characters, see the routine GPR_$ENABLE_INPUT in Chapter 11 of this manual.

GPR_$LINE_PATTERN_T specifies the bit pattern for drawing lines. This is four-element array of I*2 variables. The pattern bits are used in the following order: most significant bit of the first array element to least significant bit of the last array element.

GPR_$LINESTYLE_T specifies a line as one of two types:

```
GPR_$SOLID,
GPR_$DOTTED
```

This is an I*2 variable.

GPR_$MASK_T describes a set of planes of a bitmap. The bit corresponding to plane zero is the low-order bit of the word. If a bit of the mask is on, the corresponding plane is active. This is an I*2 variable.

GPR_$OBSCURED_OPT_T specifies the action for acquire-display when the window is obscured as one of the following:

```
GPR_$OK_IF_OBS,
GPR_$ERR_IF_OBS,
GPR_$POP_IF_OBS,
GPR_$BLOCK_IF_OBS
```

This is an I*2 variable.

GPR_$OFFSET_T describes the width and height of a bitmap or a rectangular section of a bitmap. It contains two 2-byte integers. The first designates the width (x_size); the second designates the height (y_size). In FORTRAN, use a declaration like the following:

```
      I*2 OFFSET (2)
C     (1) = X_SIZE
C     (2) = Y_SIZE
```

GPR_$PIXEL_ARRAY_T describes a list of pixel values. It contains 4-byte integers, each designating a pixel value. In FORTRAN, use a declaration like the following:

```
      I*4 PIXEL_ARRAY (n)
C     (1) = FIRST_PIXEL_VALUE
C     (2) = SECOND_PIXEL_VALUE
            .
            .
            .
```

GPR_$PIXEL_VALUE_T indicates the text color/intensity value. This is a 4-byte integer.

GPR_$POSITION_T describes an x-, y-coordinate position on the screen, a bitmap, or the cursor. It contains two 2-byte integers; the first designates the x-coordinate and the second designates the y-coordinate. In FORTRAN, use a declaration like the following:

```
      INTEGER*2 POSITION (2)
C     (1) = X_COORD
C     (2) = Y_COORD
```

GPR_$RASTER_OP_ARRAY_T describes a list of raster operation codes. It contains 2-byte integers. In FORTRAN, use a declaration like the following:

```
      INTEGER*2 RASTER_OP_ARRAY (n)
C     (1) = FIRST_RASTER_OP
C     (2) = SECOND_RASTER_OP
            .
            .
            .
```

GPR_$RASTER_OP_T specifies the value of the raster operation code. This is an I*2 variable, ranging from 0 to 15.

GPR_$STRING_T describes a character string for use in GPR_$TEXT calls. In FORTRAN, use a declaration like the following:

```
      CHARACTER STRING (256)
```

GPR_$TRAP_T describes a trapezoid by specifying its top and bottom segments in GPR_$HORIZ_SEG_T format. GPR_$TRAP_T contains six 2-byte integers. In FORTRAN, use a declaration like the following:

```
      INTEGER*2 TRAPEZOID (3,2)
C     (1,1) = TOP_LEFT_X_COORD
C     (2,1) = TOP_RIGHT_X_COORD
C     (3,1) = TOP_Y_COORD
C     (1,2) = BOTTOM_LEFT_X_COORD
C     (2,2) = BOTTOM_RIGHT_X_COORD
C     (3,2) = BOTTOM_Y_COORD
```

GPR_$TRAP_LIST_T describes a list of trapezoids in GPR_$TRAP_T format. In FORTRAN, use a declaration like the following:

```
      INTEGER*2 TRAPEZOID_LIST (3,2,n)
C     (1,1,1) = TOP_LEFT_X_COORD
C     (2,1,1) = TOP_RIGHT_X_COORD
C     (3,1,1) = TOP_Y_COORD
C     (1,2,1) = BOTTOM_LEFT_X_COORD
C     (2,2,1) = BOTTOM_RIGHT_X_COORD
C     (3,2,1) = BOTTOM_Y_COORD
C     (1,1,2) = TOP_LEFT_X_COORD
C     (2,1,2) = TOP_RIGHT_X_COORD
C     (3,1,2) = TOP_Y_COORD
C     (1,2,2) = BOTTOM_LEFT_X_COORD
C     (2,2,2) = BOTTOM_RIGHT_X_COORD
C     (3,2,2) = BOTTOM_Y_COORD
                .
                .
                .
```

GPR_$VERSION_T describes the version number on the header of an external bitmap file. This is a two-element array of two 2-byte integers: a major version number and a minor version number. Currently, both values must be 1.

GPR_$WINDOW_T describes a rectangular window by specifying both its origin (top left corner; type: GPR_$POSITION_T) and its width and height (type: GPR_$OFFSET_T). The width and height are pixel counts and include the origin pixel. (For example, a window with the origin at (10,20) and offsets (5,7) encompasses the pixels in columns 10 through 14 and rows 20 through 26, inclusive.) GPR_$WINDOW_T contains four 2-byte integers. In order, they are the start X coordinate, the start Y coordinate, the X size and the Y size. In FORTRAN, use a declaration like the following:

```
      INTEGER*2 WINDOW (2,2)
C     (1,1) = WINDOW_START_X
C     (2,1) = WINDOW_START_Y
C     (1,2) = WINDOW_OFFSET_X
C     (2,2) = WINDOW_OFFSET_Y
```

GPR_$WINDOW_LIST_T describes a list of windows in GPR_$WINDOW_T format. In FORTRAN, use a declaration like the following:

```
      INTEGER*2 WINDOW (2,2,n)
C     (1,1,1) = WINDOW_START_X
C     (2,1,1) = WINDOW_START_Y
C     (1,2,1) = WINDOW_OFFSET_X
C     (2,2,1) = WINDOW_OFFSET_Y
C     (1,1,2) = WINDOW_START_X
C     (2,1,2) = WINDOW_START_Y
C     (1,2,2) = WINDOW_OFFSET_X
C     (2,2,2) = WINDOW_OFFSET_Y
                .
                .
```

9.3 ERROR MESSAGES

The following are the possible error messages generated by the graphics primitives package.

GPR_$NOT_INITIALIZED	Primitives are not initialized.
GPR_$ALREADY_INITIALIZED	Primitives are already initialized.
GPR_$WRONG_DISPLAY_HARDWARE	The display hardware is wrong.
GPR_$ILLEGAL_FOR_FRAME	Operation is illegal for DM frame.
GPR_$MUST_BORROW_DISPLAY	You must borrow the display for this operation.
GPR_$NO_ATTRIBUTES_DEFINED	No attributes are defined for the bitmap.
GPR_$NO_MORE_SPACE	No more bitmap space is available.
GPR_$DIMENSION_TOO_BIG	The bitmap dimension is too big.
GPR_$DIMENSION_TOO_SMALL	The bitmap dimension is too small.
GPR_$BAD_BITMAP	The bitmap descriptor is incorrect.
GPR_$BAD_ATTRIBUTE_BLOCK	The attribute block descriptor is incorrect.
GPR_$WINDOW_OUT_OF_BOUNDS	Window origin is out of bitmap bounds.
GPR_$SOURCE_OUT_OF_BOUNDS	Source window origin is out of bitmap bounds.
GPR_$DEST_OUT_OF_BOUNDS	Destination window origin is out of bitmap bounds.
GPR_$INVALID_PLANE	The plane number is invalid.
GPR_$CANT_DEALLOCATE	You cannot deallocate this bitmap.
GPR_$COORD_OUT_OF_BOUNDS	Coordinate value is out of bounds.
GPR_$INVALID_COLOR_MAP	The color map is invalid.
GPR_$INVALID_RASTER_OP	The raster operation value is invalid.
GPR_$BITMAP_IS_READ_ONLY	Bitmap is read-only.
GPR_$INTERNAL_ERROR	This is an internal error.
GPR_$FONT_TABLE_FULL	Font table is full.
GPR_$BAD_FONT_FILE	Font file is incorrect.
GPR_$INVALID_FONT_ID	Font id is invalid.
GPR_$WINDOW_OBSCURED	Window is obscured.
GPR_$NOT_IN_DIRECT_MODE	Display is not in direct mode.
GPR_$NOT_IN_POLYGON	No polygon is being defined.
GPR_$KBD_NOT_ACQ	Keyboard has not been acquired.
GPR_$DISPLAY_NOT_ACQ	Display has not been acquired.
GPR_$ILLEGAL_PIXEL_VALUE	Pixel value range is illegal.
GPR_$ILLEGAL_WHEN_IMAGING	Call is illegal in imaging format.
GPR_$INVALID_IMAGING_FORMAT	Format is invalid for display hardware.
GPR_$MUST_RELEASE_DISPLAY	You must release the display for this operation.
GPR_$CANT_MIX_MODES	You cannot mix display modes, for example, borrow and direct.
GPR_$NO_INPUT_ENABLED	No input events are enabled.
GPR_$DUPLICATE_POINTS	Duplicate points are illegal.

GPR_$ARRAY_NOT_SORTED	Array must be in ascending order.
GPR_$CHARACTER_NOT_IN_FONT	Character is not in a font.
GPR_$ILLEGAL_FILL_PATTERN	Illegal bitmap for a fill pattern.
GPR_$ILLEGAL_FILL_SCALE	Fill pattern scale must be one.
GPR_$INCORRECT_ALIGNMENT	Bitmap layout specifications do not satisfy GPR alignment constraints.

CHAPTER 10

PROGRAMS AND TECHNIQUES

This chapter presents sample programs and techniques for using graphics primitives. Programs include a brief Pascal program to illustrate a basic programming format. A FORTRAN program illustrates creating a memory bitmap, drawing a figure, adding text, and selecting a display mode. Other programs and techniques include the following: the use of bit plane masks; input and event wait routines; rubber banding; and display modes.

10.1 EXAMPLE PROGRAM: PASCAL

The following program draws a three-sided, elongated box. The lines of the box are drawn with two line styles -- solid and dotted. Input, compile, and run this short program for a demonstration of a few routines.

```
program testgpr;
%nolist;
%include '/sys/ins/base.ins.pas';       {required insert file}
%include '/sys/ins/gpr.ins.pas';        {required insert file}
%list;

VAR
   st : status_$t;

   disp_bm_size : gpr_$offset_t;      {data structure for size of initial bitmap}

   init_bitmap : gpr_$bitmap_desc_t; {data structure for descriptor of
                                      initial bitmap}

   attr_desc : gpr_$attribute_desc_t; {data structure for attributes}

   i : integer;                       {type of value for line drawing}
```

```
BEGIN

{Initialization is required.  Borrow mode is chosen here for display.}

disp_bm_size.x_size := 1024;
disp_bm_size.y_size := 1024;
gpr_$init(gpr_$borrow,1,disp_bm_size,0,init_bitmap,st);


for i:= 100 to 600 do begin

   {Call gpr_$move to change the current position.}

   gpr_$move(i,i,st);

   {Draw lines from the current position to the given position.}

   gpr_$line(i+100,i,st);
   gpr_$line(i+100,i+100,st);

   {Change the linestyle from solid to dotted.}

   gpr_$set_linestyle(gpr_$dotted,5,st);

   {Draw dotted lines from the new current position to the given position.}

   gpr_$line(i,i+100,st);


   {Change the line style back to solid.}

   gpr_$set_line style(gpr_$solid,0,st);
   end;

  {End the use of graphics primitives.  }

gpr_$terminate(false,st);
END.
```

10.2 EXAMPLE PROGRAM: FORTRAN

This program creates a memory bitmap, transfers the bitmap to the display using a BLT, and adds a caption in the display. The program asks the user whether to use a frame of the Display Manager or to borrow the display. The program uses GPR_$ENABLE_INPUT. This routine requires a set of characters. For a description of the procedure for building a set of characters, see GPR_$ENABLE_INPUT in Chapter 11 of this manual.

```
C     This program must be compiled with the -I*2 option!
C
C
      PROGRAM DEMO
C
      IMPLICIT INTEGER*2 (A-Z)
C
C
C    Required insert files
C
%INCLUDE '/SYS/INS/BASE.INS.FTN'
%INCLUDE '/SYS/INS/SMDU.INS.FTN'
%INCLUDE '/SYS/INS/ERROR.INS.FTN'
%INCLUDE '/SYS/INS/GPR.INS.FTN'
C
C
      CHARACTER ANS(3)
      INTEGER*2 MODE, UNIT_OR_PAD, FONT
      INTEGER*2 BSIZE(2), SWIND(2,2), DPOS(2)
      INTEGER*2 BOX_X(4), BOX_Y(4), STAR_X(8), STAR_Y(8)
      INTEGER*4 ST, TERM_ST
      INTEGER*4 DISP_DESC, MEM_DESC, ATTR_DESC
      INTEGER*2 KEY_SET(16)
      INTEGER*2 EV_TYPE, EV_POS(2)
      CHARACTER EV_CHAR
      LOGICAL UNOBSCURED
C
C   KEY_SET is a set of all 8-bit characters.
C
      DATA KEY_SET /16 * 16#FFFF/
C
C   Describe the figure to be drawn.
C
      DATA BOX_X  / 200, 200,   0,  0/
      DATA BOX_Y  /   0, 200, 200,  0/
      DATA STAR_X / 200,   0, 100, 200,   0, 200, 100,  0/
      DATA STAR_Y / 100, 200,   0, 200, 100,   0, 200,  0/
C
C
 1010 FORMAT (' Do you want to use a Display Manager frame?')
 1020 FORMAT (3A1)
C
C   Ask which way to display the result.
```

```
C
      WRITE (*, 1010)
      READ (*, 1020) ANS
      MODE = GPR_$BORROW
      IF (ANS(1) .EQ. 'Y') MODE = GPR_$FRAME
C
C   Initialize the graphics primitives.
C
      BSIZE(1) = 800
      BSIZE(2) = 800
      UNIT_OR_PAD = 1
      CALL GPR_$INIT(MODE, UNIT_OR_PAD, BSIZE, GPR_$HIGHEST_PLANE,
     *               DISP_DESC, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Allocate an attribute block.
C
      CALL GPR_$ALLOCATE_ATTRIBUTE_BLOCK(ATTR_DESC, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Allocate a memory bitmap.
C
      BSIZE(1) = 201
      BSIZE(2) = 201
      CALL GPR_$ALLOCATE_BITMAP(BSIZE, GPR_$HIGHEST_PLANE, ATTR_DESC,
     *                          MEM_DESC, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Make it the current bitmap.
C
      CALL GPR_$SET_BITMAP(MEM_DESC, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Draw the figure in it.
C
      CALL GPR_$MOVE(0, 0, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
      CALL GPR_$POLYLINE(BOX_X, BOX_Y, 4, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
      CALL GPR_$POLYLINE(STAR_X, STAR_Y, 8, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Invert some pixels in it.
C
      CALL GPR_$SET_RASTER_OP(0, 10, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
      SWIND(1,1) = 50
      SWIND(2,1) = 50
      SWIND(1,2) = 101
      SWIND(2,2) = 101
      DPOS(1) = 50
      DPOS(2) = 50
      CALL GPR_$PIXEL_BLT(MEM_DESC, SWIND, DPOS, ST)
```

```
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Make the display the current bitmap.
C
      CALL GPR_$SET_BITMAP(DISP_DESC, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Do the bit block transfer to the display.
C
      SWIND(1,1) = 0
      SWIND(2,1) = 0
      SWIND(1,2) = 201
      SWIND(2,2) = 201
      DPOS(1) = 300
      DPOS(2) = 300
      CALL GPR_$PIXEL_BLT(MEM_DESC, SWIND, DPOS, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   Write the caption.
C
      CALL GPR_$LOAD_FONT_FILE('STD', 3, FONT, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
      CALL GPR_$SET_TEXT_FONT(FONT, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
      CALL GPR_$MOVE(375, 550, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
      CALL GPR_$TEXT('Widget', 6, ST)
      IF (ST .NE. STATUS_$OK) GOTO 9998
C
C   If display is borrowed, wait for a key to be struck.
C
      IF (MODE .EQ. GPR_$BORROW) THEN
        CALL GPR_$ENABLE_INPUT(GPR_$KEYSTROKE, KEY_SET, ST)
        IF (ST .NE. STATUS_$OK) GOTO 9998
        UNOBSCURED = GPR_$EVENT_WAIT(EV_TYPE, EV_CHAR, EV_POS, ST)
        IF (ST .NE. STATUS_$OK) GOTO 9998
      ENDIF
C
C   Finish up.
C
      CALL GPR_$TERMINATE(.FALSE., ST)
      GOTO 9999
C
C   Report an error.
C
 9998 CONTINUE
      CALL GPR_$TERMINATE(.FALSE., TERM_ST)
      CALL ERROR_$PRINT(ST)
C
C   Exit.
C
 9999 CONTINUE
      END
```

10.3 USING BIT PLANE MASKS

Masking a plane so that a program cannot write pixel data to it has a variety of uses. One use is to mask a bit plane that holds a grid. The grid remains unchanged when it is displayed with other graphic images. This technique requires writing a plane with a grid and then masking the plane so that other data is not written to it.

Another use of bit plane masks is illustrated by the program below. This program uses an eight-plane refresh buffer to hold two independent images. The program shows first one image and then the other. This is accomplished by careful loading of the color map by placing all the color values associated with each image into appropriate slots of the bitmap. The program also demonstrates the use of TIME_$WAIT to set the interval between the display of images.

```
program planes_example;

%insert '/sys/ins/base.ins.pas';
%insert '/sys/ins/gpr.ins.pas';
%insert '/sys/ins/time.ins.pas';
const
   one_second = 250000;

var
    st: status_$t;
    i: integer;

    bitm: gpr_$bitmap_desc_t;
    bitm_size: gpr_$offset_t;
    bitm_hi_plane: gpr_$plane_t;

    plane_mask: gpr_$mask_t;
    current_value: gpr_$pixel_value_t;
    color_map: gpr_$color_vector_t;
    dim_gray: gpr_$color_t;

    wait_time: time_$clock_t;

        procedure draw_triangle(x, y: gpr_$coordinate_t);

        { Draw a triangle based at x, y. }

        var
            st: status_$t;

            p1, p2, p3: gpr_$position_t;
```

```
    begin   { draw_triangle }
        p1.x_coord := x;
        p1.y_coord := y;
        p2.x_coord := x + 150;
        p2.y_coord := y + 30;
        p3.x_coord := x + 30;
        p3.y_coord := y + 150;
        gpr_$triangle(p1, p2, p3, st);
        end;    { draw_triangle }


begin   { planes_example }

    { Initialize by borrowing the display and checking the refresh buffer size.}

    bitmr_size.x_size := 1024;
    bitmr_size.y_size := 1024;
    bitmr_hi_plane := 7;
    gpr_$init(gpr_$borrow, 1, bitmr_size, bitmr_hi_plane, bitmr, st);
    gpr_$inq_bitmap_dimensions(bitmr, bitmr_size, bitmr_hi_plane, st);

    if bitmr_hi_plane = 7 then begin     { only if an eight-plane system }

        { Blank out image while you draw it. }

        for i := 0 to 255 do color_map[i] := gpr_$black;
        gpr_$set_color_map(0, 256, color_map, st);

        { First, draw a three-plane image of overlapping triangles
          in planes 0-2.}

        plane_mask := [0, 1, 2];
        gpr_$set_plane_mask(plane_mask, st);
        for i := 0 to 7 do begin        { 8 colors }
            current_value := i;
            gpr_$set_fill_value(current_value, st);
            draw_triangle(i*100, i*50);
            end;    { for }

        { Next draw a five-plane image of overlapping triangles in planes 3-7. }

        plane_mask := [3, 4, 5, 6, 7];
        gpr_$set_plane_mask(plane_mask, st);
        for i := 0 to 31 do begin       { 32 colors }
            current_value := lshft(i, 3);
            gpr_$set_fill_value(current_value, st);
            draw_triangle(i*10, i*25);
            end;    { for }

        { Set up a five-second wait time. }

        wait_time.high16 := 0;
        wait_time.low32 := 5* one_second;
```

```
    { Now show the first image and wait five seconds. }

    for i := 0 to 255 do begin      { Set up color map to show planes 0-2. }
        case (i & 7) of             { Use the eight major colors }
        0:  color_map[i] := gpr_$black;
        1:  color_map[i] := gpr_$blue;
        2:  color_map[i] := gpr_$green;
        3:  color_map[i] := gpr_$cyan;
        4:  color_map[i] := gpr_$red;
        5:  color_map[i] := gpr_$magenta;
        6:  color_map[i] := gpr_$yellow;
        7:  color_map[i] := gpr_$white;
            end;    { case }
        end;    { for }

    gpr_$set_color_map(0, 256, color_map, st);
    time_$wait(time_$relative, wait_time, st);

    { Now show the second image and wait five seconds. }

    dim_gray := 16  00080808;        { gray, 1/32 of white's intensity }
    for i := 0 to 255 do begin      { Set up color map to show planes 3-7.}
        color_map[i] := rshft(i, 3) * dim_gray; { use gray scale }
        end;    { for }
    gpr_$set_color_map(0, 256, color_map, st);
    time_$wait(time_$relative, wait_time, st);

    end;    { if bitm_hi_plane = 7 }

{ Terminate the use of graphics. }

gpr_$terminate(false, st)

end.    { planes_example }
```

10.4 USING INPUT DEVICES

This program draws diagonal vectors as it moves around a window. The user can interactively change raster operations by pressing buttons on the mouse.

```
program vectors;

%nolist;
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/gpr.ins.pas';
%include '/sys/ins/error.ins.pas';
%list;

const
    inc = 32;

var
    bitmap : gpr_$bitmap_desc_t;
    status : status_$t;
    size : gpr_$offset_t;
    line : boolean;
    loop : boolean;
    unobs : boolean;
    ev_type : gpr_$event_t;
    keystroke : char;
    position : gpr_$position_t;
    start, finish : gpr_$position_t;
    dir : (plus_x, plus_y, minus_x, minus_y);
    i : integer;
    buttons : gpr_$keyset_t := ['a','b','c','A','B','C'];

procedure check;

begin
    if status.all <> status_$ok then
        error_$print (status);
    end; { procedure check }

begin {program}

    { Initialize to maximum displayed window size. }

    size.x_size := 1024;
    size.y_size := 1024;

    { Initialize GPR in direct mode for standard output. }

    gpr_$init(gpr_$direct,stream_$stdout,size,0,bitmap,status);
    check;
```

```
{ Find out the size of the bitmap. GPR
  shrinks the bitmap to fit the window if necessary. }

gpr_$inq_bitmap_dimensions (bitmap, size, i, status);
check;
writeln ('size :', size.x_size, ' by ', size.y_size);

{ Set attributes for popping and auto-refresh. }

gpr_$set_obscured_opt (gpr_$pop_if_obs,status);
gpr_$set_auto_refresh (true, status);

{ Enable 'q' and 'Q', buttons, and locator input.  All other
  types of input go through the Display Manager. }

gpr_$enable_input(gpr_$keystroke,['Q', 'q'],status);
check;
gpr_$enable_input(gpr_$buttons, buttons, status);
check;
gpr_$enable_input(gpr_$locator, [],status);
check;
gpr_$enable_input (gpr_$locator_stop, [], status);
check;

{ Enable display of the cursor. }

gpr_$set_cursor_active (true, status);
check;

{ Initialize variables for the loop }

loop := true;
start.x_coord := 0;
start.y_coord := 0;
finish.x_coord := size.x_size - 1;
finish.y_coord := size.y_size - 1;
dir := plus_x;

{ Acquire the display and begin the loop. }

unobs := gpr_$acquire_display(status);
check;
repeat

    { Move to a new position and draw a vector. }

    gpr_$move (start.x_coord, start.y_coord, status);
    check;
    gpr_$line (finish.x_coord, finish.y_coord, status);
    check;

    { The CASE statement calculates the coordinates for the
      next vector.  The starting point for the vector is
```

```
  initially the upper left corner of the window: the
  vector moves around the window from there. For example,
  if the direction is positive for x (plus_x), the
  x-coordinate of the starting point is incremented
  by inc, the x-coordinate of the ending point is
  decremented by inc, and the y-coordinates remain
  constant. }

case dir of

plus_x : begin
   start.x_coord := start.x_coord + inc;
   finish.x_coord := size.x_size - 1 - start.x_coord;
   if (start.x_coord >= size.x_size - 1) then begin
       start.x_coord := size.x_size - 1;
       finish.x_coord := 0;
       dir := plus_y;
       end;
   end;
plus_y : begin
   start.y_coord := start.y_coord + inc;
   finish.y_coord := size.y_size - 1 - start.y_coord;
   if (start.y_coord >= size.y_size - 1) then begin
       start.y_coord := size.y_size - 1;
       finish.y_coord := 0;
       dir := minus_x;
       end;
   end;
minus_x : begin
   if (start.x_coord <= inc) then begin
       start.x_coord := 0;
       finish.x_coord := size.x_size - 1;
       dir := minus_y;
       end
   else begin
       start.x_coord := start.x_coord - inc;
       finish.x_coord := size.x_size - 1 - start.x_coord;
       end; { if then else}
   end;
minus_y : begin
   if (start.y_coord <= inc) then begin
        start.y_coord := 0;
        finish.y_coord := size.y_size - 1;
        dir := plus_x;
        end
   else begin
       start.y_coord := start.y_coord - inc;
       finish.y_coord := size.y_size - 1 - start.y_coord;
       end; {if then else}
 end;
end; {case}
```

```
    { Now look for user input. }

    unobs := gpr_$cond_event_wait (ev_type, keystroke, position, status);
    check;

    case ev_type of

    { Type 'q' or 'Q' to exit from the program. }

    gpr_$keystroke: begin
            case keystroke of
            'Q','q': loop := false;
            end; {case keystroke}
        end; {gpr_$keystroke}

    { Move the cursor. }

    gpr_$locator : begin
         gpr_$set_cursor_position(position,status);
         end;

    { Change the raster ops. }

    gpr_$buttons: begin
             case keystroke of
             'a' :  gpr_$set_raster_op (0,1,status);
             'b' :  gpr_$set_raster_op (0,6,status);
             'c' :  gpr_$set_raster_op (0,15,status);
             end; {case}
      end; {gpr_$buttons}

    end; {case ev_type}
until not loop;

{ Exit neatly; release the display and terminate the use of GPR. }

gpr_$release_display (status);
check;
gpr_$terminate(true,status);
check;
end.
```

10.5 USING RUBBER BANDING

The following program allows the user to draw lines on a monochromatic display. As a line is being placed, you are provided with feedback known as 'rubber banding'. This means that the line defined by the first end point and the current location of the cursor is continually drawn and erased as a new cursor location is obtained. This is accomplished by drawing and redrawing the same line with the 'exclusive or' raster_op set.

To begin or terminate lines, use either of the first two mouse buttons. Use the last mouse button to end the program. Alternatively, you can use the first two program function keys to begin and terminate lines and the third function key to end the program.

```
PROGRAM rubberband;

%nolist ;
%include '/sys/ins/base.ins.pas' ;
%include '/sys/ins/gpr.ins.pas' ;
%include '/sys/ins/error.ins.pas' ;
%include '/sys/ins/kbd.ins.pas' ;
%list ;

CONST
  black = 0 ;
  white = 1 ;

VAR
  offset : gpr_$offset_t ;
  pos : gpr_$position_t ;
  i : integer ;
  b_desc : gpr_$bitmap_desc_t ;
  status : status_$t ;
  size: gpr_$offset_t;
  mouse_buttons : gpr_$keyset_t := ['a', 'b', 'c'];
  pfks : gpr_$keyset_t := [kbd_$f1, kbd_$f2, kbd_$f3];
  null_buttons : gpr_$keyset_t := [ ] ;
  first : boolean ;
  et : gpr_$event_t ;
  ed : char ;
  last, anchor : gpr_$position_t ;
  hose : boolean ;
  rect : gpr_$window_t;

BEGIN
  offset.x_size := 800 ;
  offset.y_size := 800 ;
  gpr_$init (gpr_$borrow, 1, offset, 0, b_desc, status ) ;
  pos.x_coord := 400 ;
  pos.y_coord := 400 ;

  { Place a filled object for visual interest.}
```

```
rect.window_base.x_coord := 220;
rect.window_base.y_coord := 180;
rect.window_size.x_size :=  220;
rect.window_size.y_size :=  180;
gpr_$rectangle ( rect, status);

gpr_$enable_input ( gpr_$buttons, mouse_buttons, status ) ;
gpr_$enable_input ( gpr_$keystroke, pfks, status );

REPEAT
   first := true ;
   { Set 'exclusive or' raster op. }
   gpr_$set_raster_op ( 0, 6, status ) ;

   { Wait for the inital mouse key to begin. }
   gpr_$set_cursor_active ( true, status ) ;
   hose := gpr_$event_wait ( et, ed, pos, status ) ;
   gpr_$set_cursor_active ( false, status ) ;
   if ((ed = 'c') or (ed = kbd_$f3)) then exit;
   anchor.x_coord := pos.x_coord ;
   anchor.y_coord := pos.y_coord ;
   gpr_$move ( anchor.x_coord, anchor.y_coord, status ) ;

   gpr_$enable_input ( gpr_$locator, null_buttons, status ) ;

   { Rubberband to the locator position until mouse key. }

   REPEAT
       hose := gpr_$event_wait ( et, ed, pos, status ) ;
       IF et = gpr_$locator THEN begin
          IF not first THEN begin
             gpr_$move ( anchor.x_coord, anchor.y_coord, status ) ;
             gpr_$line ( last.x_coord, last.y_coord, status ) ;
             end
          ELSE
             first := false ;

          gpr_$set_draw_value ( white, status ) ;
          gpr_$move ( anchor.x_coord, anchor.y_coord, status ) ;
          gpr_$line ( pos.x_coord, pos.y_coord, status ) ;


          last.x_coord := pos.x_coord ;
          last.y_coord := pos.y_coord ;
          end ; { if locator }

       UNTIL ((et = gpr_$buttons) or (et = gpr_$keystroke));

   { Now really draw the line with normal a raster_op. }

   gpr_$set_raster_op ( 0, 3, status);
   gpr_$move ( anchor.x_coord, anchor.y_coord, status ) ;
```

```
    gpr_$line ( last.x_coord, last.y_coord, status ) ;

    gpr_$disable_input ( gpr_$locator, status ) ;
    UNTIL false;

  gpr_$terminate ( false, status ) ;

END.
```

10.6 STORING A BITMAP EXTERNALLY

The following two examples illustrate how to create of bitmap for external storage and copy it to a bitmap file. The second example shows how to access the stored bitmap and display it.

```
program dump_screen;
%nolist;
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/gpr.ins.pas';
%list;

var
    window      : gpr_$window_t      := [[0,0],[1024,1024]];
    hi_plane    : gpr_$plane_t       := 7;
    groups      : integer            := 1;
    bitmap      : gpr_$bitmap_desc_t;
    status      : status_$t;
    version     : gpr_$version_t;
    header      : gpr_$bmf_group_header_array_t;
    attribs     : gpr_$attribute_desc_t;
    filebm      : gpr_$bitmap_desc_t;
    created     : boolean;

begin
gpr_$init(gpr_$borrow_nc,1,window.window_size,hi_plane,bitmap,status);
gpr_$inq_bitmap_dimensions(bitmap,window.window_size,hi_plane,status);
header[0].n_sects          := hi_plane + 1;
header[0].pixel_size       := 1;
header[0].allocated_size   := 0;
header[0].bytes_per_line   := 0;
header[0].bytes_per_sect   := 0;
gpr_$allocate_attribute_block(attribs,status);

 {Use the access parameter gpr_$create for creating a bitmap file.}

gpr_$open_bitmap_file(gpr_$create,'screen_dump.bmf',15,
                      version,window.window_size,groups,header,
                      attribs,filebm,created,status);
```

```
gpr_$set_bitmap(filebm,status);
gpr_$pixel_blt(bitmap,window,window.window_base,status);
end.
```

This program opens the created bitmap file for display in read only mode.

```
program show_screen;
%nolist;
%include '/sys/ins/base.ins.pas';
%include '/sys/ins/gpr.ins.pas';
%list;

var
    window      : gpr_$window_t     := [[0,0],[1024,1024]];
    hi_plane    : gpr_$plane_t      := 7;
    groups      : integer;
    bitmap      : gpr_$bitmap_desc_t;
    status      : status_$t;
    version     : gpr_$version_t;
    header      : gpr_$bmf_group_header_array_t;
    attribs     : gpr_$attribute_desc_t;
    filebm      : gpr_$bitmap_desc_t;
    created     : boolean;

begin
gpr_$init(gpr_$borrow,1,window.window_size,hi_plane,bitmap,status);
gpr_$allocate_attribute_block(attribs,status);
gpr_$open_bitmap_file(gpr_$readonly,'screen_dump.bmf',15,
                      version,window.window_size,groups,header,
                      attribs,filebm,created,status);
gpr_$pixel_blt(filebm,window,window.window_base,status);
end.
```

10.7 USING DISPLAY MODES

The choice of display mode depends on which characteristics work best with your graphics operation. The four modes and their characteristics are as follows.

10.7.1 Borrow-Display Mode

Advantages

- Allows use of the entire screen
- Gives control over the entire color map.
- Provides the option of not clearing the bitmap.

Disadvantages

- Uses entire screen
- Suspends Display Manager; other processes are not immediately available.

Because of its full-screen display, borrow-display mode is useful for some applications programs. Borrow-display acquires the display hardware. This gives you control over the entire color map, including the colors otherwise reserved for the display manager. However, other processes are not available until the borrowing program returns the display.

Borrow-display mode is required for imaging formats because these formats reconfigure the entire refresh buffer.

10.7.2 Direct Mode

Advantages

- Offers the performance of borrow-display mode.
- Can use any rectangular part of the screen.
- Retains the use of the Display Manager.

Disadvantages

Does not allow control of the entire color map.

In direct mode, the program repeatedly acquires and releases the display for brief periods for graphics operations. This gives the advantages of speed of operation while preserving the Display Manager's control over display functions such as changing the window size and scrolling. Direct mode allows other processes to share the display.

10.7.3 Frame Mode

Advantages

- Reserves an area within a pad for graphics display.
- Can scroll an image out of view; redraws the image when it is pushed or popped.
- Facilitates scaling.
- Can use high-level I/O calls such as READ and WRITE
- Can leave an image in the pad for later viewing

Disadvantages

- Program executes more slowly than in borrow display mode.
- "Player piano" effect: When an image has had many changes since the last call to GPR_$CLEAR, all such changes are played back. This playing back, which occurs when the window is redrawn for any reason, may take a noticeable period of time to complete.
- Frame mode places some restrictions on GPR operations (see Section 2.2.3 and descriptions of routines in Chapter 11).

Frame mode is generally considered the easiest to use. It is appropriate for simple, noninteractive applications.

10.7.4 No-Display Mode

Advantages

- Allocates a bitmap from main memory
- Can perform graphic operations to the bitmap while bypassing the display.
- Can create bitmaps larger than the display.
- Can file the bitmap or send it to a peripheral device, such as a printer

Disadvantages

Does not display images

CHAPTER 11

USER CALLABLE ROUTINES

This chapter describes graphics primitives routines in terms of format and parameters. The routines are organized alphabetically for reference. A listing by category introduces the routines.

11.1 FUNCTIONAL LIST OF ROUTINES

Use the following listing by functional category to help you find the routines you want. Some calls are included in more than one category. To aid identification, each call is described briefly.

Initialize/Terminate Graphics Package

GPR_$INIT
: Initializes the graphics primitives package.

GPR_$TERMINATE
: Terminates the graphics primitives package.

Inquire Display Configuration

GPR_$INQ_CONFIG
: Gives the value of the current display configuration.

Establish, Delete, Identify Bitmaps

GPR_$ALLOCATE_BITMAP
: Allocates a bitmap in main memory and returns a bitmap descriptor.

GPR_$ALLOCATE_BITMAP_NC
: Allocates a bitmap in main memory without setting all the pixels in the bitmap to zero, and returns a bitmap descriptor.

GPR_$ALLOCATE_HDM_BITMAP
: Allocates a bitmap in hidden display memory.

GPR_$DEALLOCATE_BITMAP
Deallocates an allocated bitmap.

GPR_$INQ_BITMAP
Returns the descriptor of the current bitmap.

GPR_$INQ_BITMAP_DIMENSIONS
Returns the size and number of planes of a bitmap.

GPR_$INQ_WINDOW_ID
Returns the character that identifies the current bitmap's window.

GPR_$OPEN_BITMAP_FILE
Opens a file for external storage of a bitmap.

GPR_$SET_BITMAP_DIMENSIONS
Modifies the size and the number of planes of a bitmap.

GPR_$SET_WINDOW_ID
Establishes the character that identifies the current bitmap's window.

Access Bitmaps

GPR_$ENABLE_DIRECT_ACCESS
Ensures the completion of display hardware operations before the program uses the pointer to access display memory.

GPR_$INQ_BITMAP_POINTER
Returns a pointer to bitmap storage in virtual address space. This routine also returns the offset in memory from the beginning of one scan line to the next.

GPR_$INQ_BM_BIT_OFFSET
Returns the bit offset that corresponds to the left edge of a bitmap in virtual address space.

GPR_$SET_BITMAP
Establishes a bitmap as the current bitmap for subsequent operations.

GPR_$REMAP_COLOR_MEMORY
Sets the single plane of color display memory to access directly.

GPR_$REMAP_COLOR_MEMORY_1
Sets the plane of hidden color display memory. This plane is mapped at the address returned by GPR_$INQ_BITMAP_POINTER.

Establish, Delete, Identify Attribute Blocks

GPR_$ALLOCATE_ATTRIBUTE_BLOCK
Allocates a data structure that contains initial settings for bitmap attributes. The routine also returns the descriptor for the data structure.

GPR_$ATTRIBUTE_BLOCK
Returns the descriptor of the attribute block associated with the given bitmap.

GPR_$DEALLOCATE_ATTRIBUTE_BLOCK
Deallocates an attribute block allocated by GPR_$ALLOCATE_ATTRIBUTE_BLOCK.

GPR_$SET_ATTRIBUTE_BLOCK
Associates an attribute block with the current bit map.

Set Attributes

GPR_$SET_CLIP_WINDOW
Changes the clipping window for the current bitmap.

GPR_$SET_CLIPPING_ACTIVE
Enables/disables a clipping window for the current bitmap.

GPR_$SET_COORDINATE_ORIGIN
Establishes x- and y-offsets to add to all x- and y-coordinates used as input for these operations: moving the current position, line drawing, and block transfers.

GPR_$SET_DRAW_VALUE
Specifies the color/intensity value to use to draw lines.

GPR_$SET_FILL_BACKGROUND_VALUE
Specifies the color/intensity value to use for drawing the background of tile fills.

GPR_$SET_FILL_PATTERN
Specifies the fill pattern to use for the current bitmap.

GPR_$SET_FILL_VALUE
Specifies the color/intensity value to use to fill rectangles.

GPR_$SET_LINE_PATTERN
Establishes the pattern used in drawing lines.

GPR_$SET_LINESTYLE
Specifies the linestyle as solid or dashed.

GPR_$SET_PLANE_MASK
Establishes a plane mask that specifies which planes to use for subsequent write operations.

GPR_$SET_RASTER_OP
Specifies a new raster operation for both BLTs and lines.

GPR_$SET_TEXT_BACKGROUND_VALUE
Specifies the color/intensity value to use for text background.

GPR_$SET_TEXT_FONT
Establishes a new font for subsequent text operations.

GPR_$SET_TEXT_VALUE
Specifies the color/intensity value to use for writing text.

Retrieve Attributes

GPR_$INQ_CONSTRAINTS
Returns the clipping window and plane mask used for the current bitmap.

GPR_$INQ_COORDINATE_ORIGIN
Returns the x- and y-offsets added to all x- and y-coordinates used as input to move, line drawing, and BLT operations on the current bitmap.

GPR_$INQ_DRAW_VALUE
Returns the color/intensity value used for drawing lines.

GPR_$INQ_FILL_BACKGROUND_VALUE
Returns the color/intensity value used for drawing the background of tile fills.

GPR_$INQ_FILL_PATTERN
Returns the fill pattern in use for the current bitmap.

GPR_$INQ_FILL_VALUE
Returns the color/intensity value used for filling rectangles.

GPR_$INQ_LINE_PATTERN
Returns the pattern used in drawing lines.

GPR_$INQ_LINESTYLE
Returns the current line style.

GPR_$INQ_RASTER_OPS
Returns the raster operations in use for the current bitmap.

GPR_$INQ_TEXT
Returns the text font and text path used for the current bitmap.

GPR_$INQ_TEXT_OFFSET
Returns the x- and y-offsets from the top left pixel of a string to the origin of the string to be written by GPR_$TEXT to the origin of its first character. This routine also returns the x- or y-offset to the pixel which is the new current position after the text is written with GPR_$TEXT.

GPR_$INQ_TEXT_VALUES
Returns the current values of color/intensity for text and text background in the current bitmap.

Color Operations

GPR_$COLOR_ZOOM
Sets the zoom scale factor for a color display.

GPR_$SELECT_COLOR_FRAME
Selects whether frame 0 or frame 1 of the color display memory is visible.

Set/Inquire Color Map

GPR_$SET_COLOR_MAP
Establishes new values for the color map.

GPR_$INQ_COLOR_MAP
Returns the current color map values.

Block Transfer (BLT)

GPR_$ADDITIVE_BLT
Adds the single plane of any bitmap to the current bitmap.

GPR_$BIT_BLT
Performs a bit block transfer from a single plane of any bitmap to a single plane of the current bitmap.

GPR_$PIXEL_BLT
Performs a pixel block transfer from any bitmap to the current bitmap.

Set/Inquire Current Position

GPR_$MOVE
Changes the current position.

GPR_$INQ_CP
Returns the current position in the current bitmap.

Line Drawing Operations

GPR_$ARC_3P
Draws an arc from the current position, through two other points.

GPR_$CIRCLE
Draws a circle with a specified radius around a specifed center point.

GPR_$DRAW_BOX
Draws an unfilled box given two opposing corners.

GPR_$LINE
Draws a line from the current position to the given position. The routine then updates the current position to the given position.

GPR_$MULTILINE
Draws a series of disconnected lines.

GPR_$POLYLINE
Draws a series of connected lines.

GPR_$SPLINE_CUBIC_P
Draws a parametric cubic spline through the control points.

GPR_$SPLINE_CUBIC_X
Draws a cubic spline as a function of x through the control points.

GPR_$SPLINE_CUBIC_Y
Draws a cubic spline as a function of y through the control points.

Fill Operations

GPR_$CIRCLE_FILLED
Draws and fills a circle with a specified radius around a specified center point.

GPR_$RECTANGLE
Fills a rectangle.

GPR_$TRIANGLE
Fills a triangle.

GPR_$TRAPEZOID
Draws and fills a trapezoid.

GPR_$MULTITRAPEZOID
Draws and fills a list of trapezoids in the current bitmap.

GPR_$START_PGON
Defines the starting position to create a loop of edges for a polygon boundary.

GPR_$PGON_POLYLINE
Defines a series of line segments forming part of a polygon boundary.

GPR_$CLOSE_FILL_PGON
Closes and fills the currently open polygon.

GPR_$CLOSE_RETURN_PGON
Closes the currently open polygon and returns the list of trapezoids within its interior.

Text Operations

GPR_$INQ_CHARACTER_WIDTH
Returns the width of the specified character in the specified font.

GPR_$INQ_HORIZONTAL_SPACING
Returns the parameter for the width of spacing between displayed characters for the specified font.

GPR_$INQ_SPACE_SIZE
Returns the width of the space to be displayed when a character requested is not in the specified font.

GPR_$INQ_TEXT
Returns the text font and text path used for the current bitmap.

GPR_$INQ_TEXT_EXTENT
Returns the x- and y-offsets a string spans when written by GPR_$TEXT.

GPR_$INQ_TEXT_OFFSET
Returns the x- and y-offsets from the top left pixel of a string to to the origin of the string's first character. This routine also returns the x- or y- offset to the pixel which is the new current position after the text is written with GPR_$TEXT.

GPR_$INQ_TEXT_PATH
Returns the direction for writing a line of text.

GPR_$INQ_TEXT_VALUES
Returns the current values of color/intensity for text and text background in the current bitmap.

GPR_$LOAD_FONT_FILE
Loads a font from a file into the display's font storage area.

GPR_$REPLICATE_FONT
Creates and loads a modifiable copy of a font.

GPR_$SET_CHARACTER_WIDTH
Specifies the width of the specified character in the specified font.

GPR_$SET_HORIZONTAL_SPACING
Specifies the parameter for the width of spacing between displayed characters for the specified font.

GPR_$SET_SPACE_SIZE
Specifies the width of the space to be displayed when a character requested is not in the specified font.

GPR_$SET_TEXT_BACKGROUND_VALUE
Specifies the color/intensity value to use for text background.

GPR_$SET_TEXT_FONT
Establishes a new font for subsequent text operations.

GPR_$SET_TEXT_PATH
Specifies the direction for writing a line of text.

GPR_$SET_TEXT_VALUE
Specifies the color/intensity value to use for writing text.

GPR_$TEXT
Writes text in the current bitmap, beginning at the current position and proceeding in the direction specified by the most recent use of GPR_$SET_TEXT_PATH.

GPR_$UNLOAD_FONT_FILE
Unloads a font that a program has loaded with GPR_$LOAD_FONT_FILE.

Cursor Control

GPR_$INQ_CURSOR
Returns information about the cursor.

GPR_$SET_CURSOR_ACTIVE
Specifies whether the cursor is displayed.

GPR_$SET_CURSOR_ORIGIN
Defines one of the cursor's pixels as the cursor origin.

GPR_$SET_CURSOR_PATTERN
Loads a cursor pattern.

GPR_$SET_CURSOR_POSITION
Establishes a position on the screen for display of the cursor.

Clear Pixel Values

GPR_$CLEAR
Sets all pixels in the current bitmap to the given color/intensity value.

Read/Write Pixel Values

GPR_$READ_PIXELS
Reads the pixel values from a window of the current bitmap and stores the values in the pixel array.

GPR_$WRITE_PIXELS
Writes the pixel values from a pixel array into a window of the current bitmap.

Postpone Color Display Modification

GPR_$WAIT_FRAME
Waits for the current frame refresh cycle to end before executing operations that modify the color display.

Input Operations

GPR_$COND_EVENT_WAIT
Returns information abut the occurrence of any event without entering a wait state.

GPR_$DISABLE_INPUT
Disables a previously enabled event type.

GPR_$ENABLE_INPUT
Enables an event type and a selected set of keys.

GPR_$EVENT_WAIT
Waits for an event or until a time-out value is reached.

GPR_$GET_EC
Returns the address of the event count associated with a graphic event.

GPR_$SET_INPUT_SID
Specifies the input pad from which programs take graphics input.

Direct Graphics Operations

GPR_$ACQUIRE_DISPLAY
Establishes exclusive access to the display hardware and the display driver.

GPR_$FORCE_RELEASE
Releases the display regardless of how many times is has been acquired.

GPR_$INQ_VIS_LIST
Returns a list of visible sections of an obsured window.

GPR_$RELEASE_DISPLAY
Decrements a counter associated with the number of times a display has been acquired.

GPR_$SET_ACQ_TIME_OUT
Establishes the length of time for which the program can acquire the display.

GPR_$SET_AUTO_REFRESH
Directs the Display Manager to refresh the window automatically.

GPR_$SET_OBSCURED_OPT
Establishes the action the program is to take when a window to be acquired is obscured.

GPR_$SET_REFRESH_ENTRY
Specifies the entry points of these application-supplied procedures: refreshing the displayed image in a direct window and refreshing the hidden display memory.

Set, Inquire Imaging Format

GPR_$SET_IMAGING_FORMAT
Sets the imaging format of the color display.

GPR_$INQ_IMAGING_FORMAT
Returns the current imaging format of the color display.

GPR_$SET_COLOR_MAP
Establishes new values for the color map.

11.2 DESCRIPTION OF ROUTINES

The remainder of this chapter describes graphics primitives routines. The descriptions are organized alphabetically for reference.

GPR_$ACQUIRE_DISPLAY -- Establishes exclusive access to the display hardware and the display driver.

FORMAT

```
unobscured := GPR_$ACQUIRE_DISPLAY (status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

unobscured
: A Boolean value that indicates whether or not the window is obscured (false=obscured). This parameter is always true unless the option GPR_$OK_IF_OBS was specified to GPR_$SET_OBSCURED_OPT.

status
: Completion status, in STATUS_$T format. A nonzero status indicates that the display was not acquired.

NOTES

* While the display is acquired, the Display Manager cannot run. Hence, it cannot respond to pad calls or to stream calls to input or transcript pads. If you need to call any of these routines, you must release the display to do so.

* Since no other display output can occur while the display is acquired, it is not a good idea to acquire the display for long periods of time. The acquire routine automatically times out after a default period of one minute; programs can change this time-out with the routine GPR_$SET_ACQ_TIME_OUT.

* Although this call is needed only in direct mode, it can be called from any of the other display modes, where it performs no operation and returns the status code GPR_$NOT_IN_DIRECT_MODE.

* If the display is already acquired when this call is made, a count of calls is incremented such that pairs of acquire/release display calls can be nested.

GPR_$ADDITIVE_BLT — Adds a single plane of any bitmap to the current bitmap.

FORMAT

```
GPR_$ADDITIVE_BLT (source-bitmap-desc, source-window, source-plane,
                   dest-origin, status)
```

INPUT PARAMETERS

source-bitmap-desc
: Descriptor of the source bitmap which contains the source window to be transferred, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

source-window
: Rectangular section of the bitmap from which to transfer pixels, in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

  The first element of GPR_$WINDOW_T specifies the window start origin (top left corner), in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the source bitmap.

  The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units.

source-plane
: The identifier of the source plane to add, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are in the range 0 through the identifier of the source bitmap's highest plane.

dest-origin
: Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the current bitmap, unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Both the source and destination bitmaps can be in either display memory or main memory.

* The source window origin is added to the coordinate origin for the source bitmap, and the result is the actual origin of the source rectangle for the BLT. Similarly, the destination origin is added to the coordinate origin for the current bitmap, and the result is the actual origin of the destination rectangle for the BLT.

LIMITATIONS

* If the source bitmap is a Display Manager frame, the only allowed raster op codes are 0, 5, A, and F. These are the raster operations in which the source plays no role.

* If a rectangle is transferred by a BLT to a display manager frame and the frame is refreshed for any reason, the BLT is re-executed. Therefore, if the information in the source bitmap has changed, the appearance of the frame changes accordingly.

GPR_$ALLOCATE_ATTRIBUTE_BLOCK — Allocates a data structure that contains initial bitmap attribute settings, and returns the descriptor for the data structure.

FORMAT

```
GPR_$ALLOCATE_ATTRIBUTE_BLOCK (attrib-block-desc, status)
```

INPUT PARAMETERS

None

OUTPUT PARAMETERS

attrib-block-desc
: Attribute block descriptor, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* When allocated, the attribute block is initialized to default settings. See Section 3.7.2 for the default settings.
* To associate an attribute block with the current bitmap, use GPR_$SET_ATTRIBUTE_BLOCK.
* To deallocate an attribute block, use GPR_$DEALLOCATE_ATTRIBUTE_BLOCK.

GPR_$ALLOCATE_BITMAP — Allocates a bitmap in main memory and returns a bitmap descriptor.

FORMAT

GPR_$ALLOCATE_BITMAP (size, hi-plane-id, attrib-block-desc, bitmap-desc, status)

INPUT PARAMETERS

size
: Bitmap width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is bitmap width, in raster units; the second element is the bitmap height, in raster units. Possible values for x are 1-4096; possible values for y are 1-4096.

hi-plane-id
: Identifier of the highest plane which the bitmap will use, in GPR_$PLANE_T format. This is a 4-byte integer. Valid values are 0-7.

attrib-block-desc
: Descriptor of the attribute block which the bitmap will use, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap-desc
: Descriptor of the allocated bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* To deallocate a bitmap, use GPR_$DEALLOCATE_BITMAP.
* A program can not use a bitmap after it is deallocated.
* To establish an allocated bitmap as the current bitmap, use GPR_$SET_BITMAP.

GPR_$ALLOCATE_BITMAP_NC — Allocates a bitmap in main memory without setting all the pixels in the bitmap to zero, and returns a bitmap descriptor.

FORMAT

GPR_$ALLOCATE_BITMAP_NC (size, hi-plane-id, attrib-block-desc, bitmap-desc, stat

INPUT PARAMETERS

size
Bitmap width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is bitmap width, in raster units; the second element is the bitmap height, in raster units. Possible values for x are 1-4096; possible values for y are 1-4096.

hi-plane-id
Identifier of the highest plane which the bitmap will use, in GPR_$PLANE_T format. This is a 4-byte integer. Valid values are 0-7.

attrib-block-desc
Descriptor of the attribute block which the bitmap will use, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap-desc
Descriptor of the allocated bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format.

NOTES

* To deallocate a bitmap, use GPR_$DEALLOCATE_BITMAP.

* A program can not use a bitmap after it is deallocated.

* To establish an allocated bitmap as the current bitmap, use GPR_$SET_BITMAP.

* GPR_$ALLOCATE_BITMAP sets all pixels in the bitmap to zero; this routine does not. As a result, GPR_$ALLOCATE_BITMAP_NC executes faster, but the initial contents of the bitmap are unpredictable.

GPR_$ALLOCATE_HDM_BITMAP -- Allocates a bitmap in hidden display memory.

FORMAT

```
GPR_$ALLOCATE_HDM_BITMAP (size, hi-plane-id, attrib-block-desc, bitmap-desc,
                          status)
```

INPUT PARAMETERS

size
: The width and height of the bitmap, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the bitmap width, in raster units; the second element is the bitmap height, in raster units.

hi-plane-id
: The identifier of the highest plane of the bitmap, in GPR_$PLANE_T format. This is a 2-byte integer.

attrib-block-desc
: The descriptor of the bitmap's attribute block, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap-desc
: The descriptor of the bitmap in hidden display memeory, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$ALLOCATE_HDM_BITMAP allocates a GPR bitmap in hidden display memory for programs in borrow-display or direct mode. In frame mode, hidden display memory bitmaps cannot be used.
* In direct mode you must acquire the display before calling GPR_$ALLOCATE_BITMAP.
* The maximum size allowed for hidden display memory bitmaps is 224 bits by 224 bits.
* Use GPR_$DEALLOCATE_BITMAP to deallocate a hidden display bitmap.

GPR_$ARC_3P — Draws an arc from the current position through two other specified points.

FORMAT

GPR_$ARC_3P (point-2, point-3, status)

INPUT PARAMETERS

point-2
: The second point on the arc, in GPR_$POSITION_$T format. This is a two-element array of 2-byte integers describing an (x, y) coordinate position in the bitmap. Its values must be within bitmap limits unless clipping is enabled.

point-3
: The third point on the arc, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers, describing an (x, y) coordinate position in the bitmap. Its values must be within bitmap limits unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the arc is drawn.
* After the arc is drawn, point-3 becomes the current position.
* An error is returned if any of the three points are equal.
* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$ATTRIBUTE_BLOCK — Returns the descriptor of the attribute block associated with the given bitmap.

FORMAT

```
attrib-block-desc = GPR_$ATTRIBUTE_BLOCK (bitmap-desc, status)
```

INPUT PARAMETERS

bitmap-desc
: Descriptor of the bitmap that is using the requested attribute block, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

RETURNED FUNCTION VALUE

attrib-block-desc
: Descriptor of the attribute block used for the given bitmap, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

NOTES

* To set an attribute block as the block for the current bitmap, use GPR_$SET_ATTRIBUTE_BLOCK.

GPR_$BIT_BLT -- Performs a bit block transfer from a single plane of any bitmap to a single plane of the current bitmap.

FORMAT

```
GPR_$BIT_BLT (source-bitmap-desc, source-window, source-plane,
              dest-origin, dest-plane, status)
```

INPUT PARAMETERS

source-bitmap-desc
: Descriptor of the source bitmap which contains the source window to be transferred, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

source-window
: Rectangular section of the bitmap from which to transfer pixels, in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

  The first element of GPR_$WINDOW_T specifies the window start origin (upper top corner), in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the source bitmap.

  The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units.

source-plane
: Identifier of the single plane of the source bitmap to move, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are in the range 0 through the identifier of the source bitmap's highest plane.

dest-origin
: Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This is a two-element array of two-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the current bitmap, unless clipping is enabled.

dest-plane
: Identifier of the plane of the destination bitmap, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are in the range 0 through the identifier of the destination bitmap's highest plane.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* Both the source and destination bitmaps can be in either display memory or main memory.

* The source window origin is added to the coordinate origin for the source bitmap, and the result is the actual origin of the source rectangle for the BLT. Similarly, the destination origin is added to the coordinate origin for the current bitmap, and the result is the actual origin of the destination rectangle for the BLT.

LIMITATIONS

* If the source bitmap is a Display Manager frame, the only allowed raster op codes are 0, 5, A, and F. These are the raster operations in which the source plays no role.

* If a rectangle is transferred by a BLT to a Display Manager frame and the frame is refreshed for any reason, the BLT is re-executed. Therefore, if the information in the source bitmap has changed, the appearance of the frame changes accordingly.

GPR_$CIRCLE — Draws a circle with the specified radius around the specified center point.

FORMAT

```
GPR_$CIRCLE (center, radius, status)
```

INPUT PARAMETERS

center
: The center of the circle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers, describing an (x, y) coordinate position in the bitmap. Its values must be within bitmap limits unless clipping is enabled.

radius
: The radius of the circle. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the center of the circle.
* The current position is not changed by use of GPR_$CIRCLE.
* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$CIRCLE_FILLED — Draws and fills a circle with the specified radius around the specified center point.

FORMAT

```
GPR_$CIRCLE_FILLED (center, radius, status)
```

INPUT PARAMETERS

center
: The center of the circle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers describing an (x, y) coordinate position in the bitmap. Its values must be within bitmap limits unless clipping is enabled.

radius
: The radius of the circle. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the center of the circle.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$CLEAR -- Sets all pixels in the current bitmap to the given color/intensity value.

FORMAT

GPR_$CLEAR (index, status)

INPUT PARAMETERS

index
New color map index specifying a color/intensity value for all pixels in the current bitmap, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0-1 for monochromatic displays
0-15 for color displays in 4-bit pixel format
0-255 for color displays in 8-bit or 24-bit pixel format
-2 for all displays.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* A special case occurs if the specified index is -2. A value of -2 specifies clearing the bitmap to the current background color/intensity value. For memory bitmaps and borrowed displays, the background color/intensity index is zero. For Display Manager frames, the background color/intensity value is the same as that used for the window background color.

* For monochromatic displays, only the low-order bit of the color value is considered, because bitmaps currently have only one plane. For color displays in 4-bit pixel mode, only the four lowest-order bits of the color value are considered because these displays have four planes.

* You can use GPR_$SET_COLOR_MAP to establish the correspondence between color map indexes and color/intensity values. This means that you can use GPR_$SET_COLOR_MAP to assign the pixel value 0 to bright intensity, and then use GPR_$CLEAR either to make the screen bright by passing the pixel value 0, or make the screen dark by passing the value 1.

* This routine is subject to the restrictions of the current clipping window and plane mask.

* The color specification parameter is a color map index, not a color value. See Section 4.3.

GPR_$CLOSE_FILL_PGON -- Closes and fills the currently open polygon.

FORMAT

GPR_$CLOSE_FILL_PGON (status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$CLOSE_FILL_PGON closes and fills the series of polygon boundaries created with the routines GPR_$START_PGON and GPR_$PGON_POLYLINE.

* GPR_$CLOSE_FILL_PGON does not use the current raster operation setting.

GPR_$CLOSE_RETURN_PGON — Closes the currently open polygon and returns the list of trapezoids within its interior.

FORMAT

GPR_$CLOSE_RETURN_PGON (list-size, trapezoid-list, trapezoid-number, status)

INPUT PARAMETERS

list-size
: The maximum number of trapezoids that the routine is to return. This is a 2-byte integer.

OUTPUT PARAMETERS

trapezoid-list
: The trapezoids returned. This is an array of trapezoids in GPR_$TRAP_T format, which is a two-element array in GPR_$HORIZ_SEG_T format. The first element is the top horizontal segment of a trapezoid; the second element is the bottom horizontal segment.

trapezoid-number
: The number of trapezoids that exist within the polygon interior. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$CLOSE_RETURN_PGON returns a list of trapezoids within a polygon interior that the graphics program can draw at a later time with the routine GPR_$MULTITRAPEZOID.

* The trapezoid-number parameter is always the total number of trapezoids composing the polygon interior. If this number is greater than the list-size parameter, some trapezoids were left out of the trapezoid-list for lack of space.

GPR_$COLOR_ZOOM — Sets the zoom scale factor for a color display.

FORMAT

GPR_$COLOR_ZOOM (xfactor, yfactor, status)

INPUT PARAMETERS

xfactor
A 2-byte integer that denotes the scale factor for the x-coordinate, in the range 1 through 15.

yfactor
A 2-byte integer that denotes the scale factor for the y-coordinate, in the range 1 through 15.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* If the x value is not equal to 1, then the y value must be not equal to 1.
* GPR_$COLOR_ZOOM uses the integer zoom feature of the color hardware.
* GPR_$COLOR_ZOOM works only in borrow-display mode.
* GPR_$COLOR_ZOOM always zooms from the upper-left corner of the display.

GPR_$COND_EVENT_WAIT — Returns information about the occurrence of any event without entering a wait state.

FORMAT

unobscured := GPR_$COND_EVENT_WAIT (event-type, event-data, position, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

event_type
The type of event that occurred, in GPR_$EVENT_T format. The possible events are:

GPR_$KEYSTROKE	Input from a keyboard
GPR_$BUTTONS	Input from mouse or bitpad puck buttons
GPR_$LOCATOR	Input from a touchpad or mouse
GPR_$ENTERED_WINDOW	Cursor has entered window
GPR_$LEFT_WINDOW	Cursor has left window
GPR_$LOCATOR_STOP	Input from a locator has stopped
GPR_$NO_EVENT	No event has occurred

event_data
The keystroke or button character associated with the event, or the character that identifies the window associated with an entered window event. This parameter is not modified for other events. In FORTRAN, this is declared as a character*1 variable.

position
The position on the screen or within the window at which graphics input occurred, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers.

unobscured
A Boolean value that indicates whether or not the window is obscured; a false value means that the window is obscured. This value is always true unless the program has called GPR_$SET_OBSCURED_OPT and specified an option of GPR_$OK_IF_OBS.

status
Completion status, in STATUS_$T format.

NOTES

* When called, this routine returns immediately and reports information about any event that has occurred. Typically, this routine is called following return from an EC2_$WAIT call involving the eventcount returned by GPR_$GET_EC. The routine allows the program to obtain information about an event without having to suspend all of its activities.

* Unless locator data has been processed since the last event was reported, "position" will be the last position given to GPR_$SET_CURSOR_POSITION.

* If locator data is received during this call, and GPR_$LOCATOR events are not enabled, the GPR software will display the arrow cursor and will set the keyboard cursor position.

* Although this call never waits, it may release the display if it receives an unenabled event that needs to be handled by the Display Manager.

* The input routines report button events as ASCII characters. "Down" transitions range from "a" to "d"; "up" transitions range from "A" to "D". The three mouse keys start with (a/A) on the left side. As with keystroke events, button events can be selectively enabled by specifying a button keyset.

GPR_$DEALLOCATE_ATTRIBUTE_BLOCK -- Deallocates an attribute block allocated by GPR_$ALLOCATE_ATTRIBUTE_BLOCK.

FORMAT

```
GPR_$DEALLOCATE_ATTRIBUTE_BLOCK (attrib-block-desc, status)
```

INPUT PARAMETERS

attrib-block-desc
: The descriptor of the attribute block to deallocate, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* To allocate an attribute block, use GPR_$ALLOCATE_ATTRIBUTE_BLOCK.
* To associate an attribute block with the current bitmap, use GPR_$SET_ATTRIBUTE_BLOCK.

GPR_$DEALLOCATE_BITMAP — Deallocates an allocated bitmap.

FORMAT

GPR_$DEALLOCATE_BITMAP (bitmap-desc, status)

INPUT PARAMETERS

bitmap-desc
    Descriptor of the bitmap to deallocate, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
    Completion status, in STATUS_$T format.

NOTES

* To allocate a bitmap, use GPR_$ALLOCATE_BITMAP or GPR_$ALLOCATE_HDM_BITMAP.

GPR_$DISABLE_INPUT — Disables a previously enabled event type.

FORMAT

GPR_$DISABLE_INPUT (event_type, status)

INPUT PARAMETERS

event_type
The type of event to be disabled, in GPR_$EVENT_T format. The types of events are:

GPR_$KEYSTROKE	Input from a keyboard
GPR_$BUTTONS	Input from mouse or bitpad puck buttons
GPR_$LOCATOR	Input from a touchpad or mouse
GPR_$ENTERED_WINDOW	Cursor has entered window
GPR_$LEFT_WINDOW	Cursor has left window
GPR_$LOCATOR_STOP	Input from a locator has stopped

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* Following this call, no events of the given event type will be returned by GPR_$EVENT_WAIT or GPR_$COND_EVENT_WAIT.
* In borrow-display mode, disabled events received by the GPR software will be ignored.
* In direct mode or frame mode, disabled keystroke or button events are processed by the Display Manager.
* When locator events are disabled, the GPR software will display the arrow cursor and will set the keyboard cursor position when locator data is received.

GPR_$DRAW_BOX — Draws an unfilled box based on the coordinates of two opposing corners.

FORMAT

```
GPR_$DRAW_BOX (corner-1, corner-2, status)
```

INPUT PARAMETERS

corner-1
: The corner of the box, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers describing an (x, y) coordinate position in the bitmap. Its values must be within bitmap limits unless clipping is enabled.

corner-2
: An opposing corner of the box, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers, describing an (x, y) coordinate position in the bitmap. Its values must be within bitmap limits unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the opposing corners of the box.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$ENABLE_DIRECT_ACCESS — Ensures completion of display hardware operations before program uses pointer to access display memory.

FORMAT

```
GPR_$ENABLE_DIRECT_ACCESS (status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* If a program uses the GPR_$INQ_BITMAP_POINTER to get the address of display memory for a monochromatic or color display, it should call GPR_$ENABLE_DIRECT_ACCESS after any calls that change the display and before using the pointer returned from the GPR_$INQ_BITMAP_POINTER.

GPR_$ENABLE_INPUT -- Enables an event type and a selected set of keys.

FORMAT

GPR_$ENABLE_INPUT (event_type, key-set, status)

INPUT PARAMETERS

event_type
The type of event to be enabled, in GPR_$EVENT_T format. The types of events are:

GPR_$KEYSTROKE	Input from a keyboard
GPR_$BUTTONS	Input from mouse or bitpad puck buttons
GPR_$LOCATOR	Input from a touchpad or mouse
GPR_$ENTERED_WINDOW	Cursor has entered window
GPR_$LEFT_WINDOW	Cursor has left window
GPR_$LOCATOR_STOP	Input from a locator has stopped

key-set
The set of specifically enabled characters when the event class is in GPR_$KEYSET_T format. This parameter is specified for event types of GPR_$KEYSTROKE and GPR_$BUTTONS.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* This routine specifies the type of event and event input for which GPR_$EVENT_WAIT is to wait.

* This routine applies to the current bitmap. However, enabled input events are stored in attribute blocks (not with bitmaps) in much the same way as attributes are. When a program changes attribute blocks for a bitmap during a graphics session, the input events you enabled are lost unless you enable those events for the new attribute block.

* Programs must call this routine once for each event type to be enabled.

* No event types are enabled by default.

* The keyset must correspond to the specified event type. For example, use [' '..'~'] (in Pascal) to enable all normal printing graphics. Use [chr(0)..chr(127)] to enable the entire ASCII character set. Except in borrow-display mode, it is a good idea to leave at least the CMD and

NEXT_WINDOW keys out of the keyset so that the user can access other Display Manager windows.

* The insert file /SYS/INS/KBD.INS contains definitions for the non-ASCII keyboard keys in the range 128..255.

* Events and keyset data not enabled with this routine will be handled by the Display Manager in frame or direct mode and discarded in borrow-display mode.

* When locator events are disabled, the GPR software will display the arrow cursor and will set the keyboard cursor position when locator data is received.

* GPR_$ENABLE_INPUT expects a Pascal set of characters as one input argument. The following subroutine provides a way to build a set of characters for a FORTRAN program using this call.

BUILD_SET — Builds a Pascal set of characters for FORTRAN users.

INPUT ARGUMENTS

list — An integer*2 array, up to 256 entries long. This array contains the ordinal values of the characters to be included in the set. For example, if you wish to include the capital letters A through Z, make the array 26 entries long, including the values 65 through 90.

no_of_entries — The number of entries used in list. An integer*2 scalar.

OUTPUT ARGUMENTS

returned_set — The equivalent of the Pascal set of characters. This can be of any type, as long as it is 32 bytes long. Use integer*4 returned_set(8).

This program does not check for errors. Therefore, values can be outside the range 0 to 255, although this can give unpredictable results. The program does not check to see if the value has already appeared in the list.

The subroutine builds the set anew each time; it does not allow you to add new elements to an existing set.

The following program builds a set of characters for FORTRAN users.

```
 PROGRAM build_set
```

```
      subroutine build_set(list,no_of_entries,returned_set)

      integer*2 list(1),no_of_entries,returned_set(0:15)
      integer*2 i,mask(0:15),word,bit
      data mask/1,2,4,8,16#10,16#20,16#40,16#80,16#100,16#200,
     1   16#400,16#800,16#1000,16#2000,16#4000,16#8000/

c     A Pascal set of characters is a 256-bit "array." The bit
c     corresponding to the ordinal position of the character is
c     1 if the bit is in the set and 0 if the character is absent
c     from the set. In this example, the set is initialized
c     to 0, that is, no characters are present.

      do 100 i=0,15
         returned_set(i) = 0
100      continue
c
c     Go through the list, setting the bits for each character listed.
c     Note that Pascal numbers the bits right to left.
c     Therefore, a set containing only char(0), that is NULL, has
c     only the least-significant bit set in the last word of the set.

      do 110 i=1,no_of_entries
c
c     Set the appropriate bit.

         word = 15 - (list(i)/16)
         bit = mod(list(i),16)
         returned_set(word) = or(returned_set(word),mask(bit))
110   continue
c
      return
      end
```

GPR_$EVENT_WAIT — Waits for an event.

FORMAT

unobscured := GPR_$EVENT_WAIT (event_type, event_data, position, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

event_type
The type of event that occurred, in GPR_$EVENT_T format. Possible values are:

GPR_$KEYSTROKE	Input from a keyboard
GPR_$BUTTONS	Input from mouse or bitpad puck buttons
GPR_$LOCATOR	Input from a touchpad or mouse
GPR_$ENTERED_WINDOW	Cursor has entered window
GPR_$LEFT_WINDOW	Cursor has left window
GPR_$LOCATOR_STOP	Input from a locator has stopped
GPR_$NO_EVENT	No event has occurred

event_data
The keystroke or button character associated with the event, or the character that identifies the window associated with an entered window event. This parameter is not modified for other events. In FORTRAN, this is declared as a character*1 variable.

position
The position on the screen or within the window at which graphics input occurred, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers.

unobscured
A Boolean value that indicates whether or not the window is obscured; a false value means that the window is obscured. This value is always true unless the program has called GPR_$SET_OBSCURED_OPT and specified an option of GPR_$OK_IF_OBS.

status
Completion status, in STATUS_$T format.

NOTES

* This routine suspends process execution until the occurrence of an event type enabled with the GPR_$ENABLE_INPUT. If the event type is keystroke or button, this routine reports only characters in the enabled keyset. Input routines report button events as ASCII characters. See Appendix for keyboard charts showing the ASCII values.

* In direct mode, time-out values do not apply to calls to GPR_$EVENT_WAIT; that is, GPR_$EVENT_WAIT waits indefinitely.

* The input routines report button events as ASCII characters. "Down" transitions range from "a" to "d"; "up" transitions range from "A" to "D". The three mouse keys start with (a/A) on the left side. As with keystroke events, button events can be selectively enabled by specifying a button keyset.

* Unless locator data has been processed since the last event was reported, "position" will be the last position given to GPR_$SET_CURSOR_POSITION.

* If locator data is received during this call, and GPR_$LOCATOR events are not enabled, the GPR software will display the arrow cursor and will set the keyboard cursor position.

* GPR_$EVENT_WAIT returns an error if the display has not previously been acquired.

* This routine may implicitly release the display under two conditions:

  The process must actually suspend execution to wait for an event.

  An event that is not enabled must be handled by the Display Manager.

GPR_$FORCE_RELEASE — Releases the display regardless of how many times it has previously been acquired.

FORMAT

```
GPR_$FORCE_RELEASE (acquire-count, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

acquire-count
: The number of times the display has been acquired. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* This call releases the display regardless of how many times GPR_$ACQUIRE_DISPLAY has been called.

GPR_$GET_EC — Returns the eventcount associated with a graphic event.

FORMAT

GPR_$GET_EC (gpr-key, eventcount-pointer, status)

INPUT PARAMETERS

gpr-key
: The key that specifies which eventcount to obtain, in GPR_$EC_KEY_T format. Currently, this key is always GPR_$INPUT_EC.

OUTPUT PARAMETERS

eventcount-pointer
: A pointer to the eventcount for graphics input, in EC2_$PTR_T format.

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$GET_EC returns the eventcount pointer for the graphics input eventcount, which is advanced whenever graphics input may be available.

* When this eventcount is advanced, it does not guarantee that GPR_$COND_EVENT_WAIT will return an event, or that GPR_$EVENT_WAIT will not wait. The advance is merely an optimization of a simple polling loop that suspends execution of the process until an event might be available.

GPR_$INIT — Initializes the graphics primitives package.

FORMAT

GPR_$INIT (op-mode, unit, size, hi-plane-id, init-bitmap-desc, status)

INPUT PARAMETERS

op-mode
One of four modes of operation. Graphics primitives routines can operate in two borrow-display modes, within a Display Manager window, within a frame of a Display Manager pad, or without using the display. Use GPR_$DISPLAY_MODE_T format for this parameter. Possible values for this parameter are:

GPR_$BORROW
GPR_$BORROW_NC
GPR_$DIRECT
GPR_$FRAME
GPR_$NO_DISPLAY

unit
This parameter has three possible meanings, as follows:

1) The display unit, if the graphics routines are to operate in a borrowed display. This is a 2-byte integer. Currently, the only valid display unit number for borrow-display mode is 1.

2) The stream identifier for the pad, if the graphics routines are to operate in frame or direct mode. Use STREAM_$ID_T format. This is a 2-byte integer.

3) Any value, such as zero, if the graphics routines do not use the display.

size
The size of the initial bitmap (and the size of the frame, in frame mode), in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the bitmap width (and frame width, in frame mode), in raster units; the second element is the bitmap height (and frame height, in frame mode), in raster units. Possible values are listed on the next page.

	X	Y
Borrow-display or direct mode (limits are reduced to display or window size if necessary):	1 to 1024	1 to 1024
Display Manager Frame:	1 to 32768	1 to 32768
Main Memory Bitmap:	1 to 4096	1 to 4096

hi-plane-id
Identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are:

For display memory bitmaps:

0 for monochromatic displays
0-3 for color displays in two-board configuration
0-7 for color displays in three-board configuration

For main memory bitmaps:

0-7 for all displays

OUTPUT PARAMETERS

init-bitmap-desc
Descriptor of the initial bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer that uniquely identifies the bitmap.

status
Completion status, in STATUS_$T format.

NOTES

* See Section 2.2 for information about the modes of operation.
* To use multiple windows, you must call GPR_$INIT for each window.
* GPR_$BORROW_NC allows you to allocate a bitmap in display memory without setting all the pixels to zero.
* In GPR_$NO_DISPLAY mode, the program can manipulate only main memory bitmaps.

* Size: If a program executes in borrow-display mode or direct mode, the size of the initial bitmap can be equal to or smaller than the display. See Section 3.3 for details. If the program executes in a frame of a Display Manager pad, "size" specifies the size of both the frame and the initial bitmap. (In frame mode, the frame and the bitmap must be the same size.) If the program does not use the display, GPR_$INIT creates a bitmap in main memory. The program specifies the size of this bitmap.

* To use imaging formats, a program must initiate a borrow-display mode.

GPR_$INQ_BITMAP — Returns the descriptor of the current bitmap.

FORMAT

GPR_$INQ_BITMAP (bitmap-desc, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

bitmap-desc
The descriptor of the current bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

status
Completion status, in STATUS_$T format.

NOTES

* To establish a bitmap as the current bitmap, use GPR_$SET_BITMAP.

GPR_$INQ_BITMAP_DIMENSIONS -- Returns the size and number of planes of a bitmap.

FORMAT

GPR_$INQ_BITMAP_DIMENSIONS (bitmap-desc, size, hi-plane-id, status)

INPUT PARAMETERS

bitmap-desc
: The descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

size
: Width and height of the bitmap, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the bitmap width, in raster units; the second element is the bitmap height, in raster units.

hi-plane-id
: The identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* A program can use the information returned by this call to retrieve the actual bitmap size. This could be useful, for example, if the program specified a bitmap size that was too large for the display, causing a reduction in bitmap size.

GPR_$INQ_BITMAP_POINTER — Returns a pointer to bitmap storage in virtual address space. Also returns offset in memory from beginning of one scan line to the next.

FORMAT

GPR_$INQ_BITMAP_POINTER (bitmap-desc, storage-ptr, storage-line-width, status)

INPUT PARAMETERS

bitmap-desc
: Descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

storage-ptr
: Start address of bitmap in virtual address space. This is a 4-byte integer. In Pascal, storage-ptr is UNIV_PTR.

storage-line-width
: Number of 16-bit words in virtual memory between the beginning of one of the bitmap's scan lines and the next. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* See Section 3.5 for more information about this routine.
* A program can use the information returned by this call to access individual bits.
* Each scan line (horizontal line of a bitmap) starts on a word boundary. "Line-width" gives the offset in memory from the beginning of one scan line to the beginning of the next, in units of 16-bit words.
* When a program uses the "storage-pointer" to access the borrowed display, pixels which are white have the value 1 and pixels which are black have the value 0, regardless of any calls to GPR_$SET_COLOR_MAP. Also, if the cursor is active, the cursor pattern appears in the bitmap.

LIMITATIONS

* A program cannot use this routine on a bitmap which is a Display Manager frame.

GPR_$INQ_BM_BIT_OFFSET — Returns the bit offset that corresponds to the left edge of a bitmap in virtual address space.

FORMAT

```
GPR_$INQ_BM_BIT_OFFSET (bitmap-desc, offset, status)
```

INPUT PARAMETERS

bitmap-desc
: The descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

offset
: The number of bits between a 16-bit word boundary and the left edge of the specified bitmap. This is a 2-byte integer in the range zero to fifteen.

status
: Completion status, in STATUS_$T format.

NOTES

* Each scan line (horizontal line of a bitmap) starts on a word boundary. For all scan lines, this routine returns the number of bits in the most significant part of the first word that are not part of the specified bitmap.

* Currently, the offset will be zero for any bitmap other than a direct-mode window.

* For more information about bitmaps, see Section 3.5.

GPR_$INQ_CHARACTER_WIDTH — Returns the width of the specified character in the specified font.

FORMAT

```
GPR_$INQ_CHARACTER_WIDTH (font-id, character, width, status)
```

INPUT PARAMETERS

font-id
: Identifier of the text font. This is a 2-byte integer.

character
: The specified character. This is a CHAR variable.

OUTPUT PARAMETERS

width
: The width parameter of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a character's width, use GPR_$SET_CHARACTER_WIDTH.
* The initial character widths are defined in the font file.
* This routine returns the character width in the local copy of the font. Initially, this is a copy of the font file; however, the local copy may have been changed. Change in the local copy does not affect the font file or the use of the font by other processes.

GPR_$INQ_COLOR_MAP — Returns the current color map values.

FORMAT

GPR_$INQ_COLOR_MAP (start-index, n-entries, values, status)

INPUT PARAMETERS

start-index
: Index of the first color value entry, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

n-entries
: Number of entries. This is a 2-byte integer.

OUTPUT PARAMETERS

values
: Color value entries, in GPR_$COLOR_VECTOR_T format. This is an array of 4-byte integers.

status
: Completion status, in STATUS_$T format.

NOTES

* To set the color map, use GPR_$SET_COLOR_MAP.

GPR_$INQ_CONFIG -- Returns the current display configuration.

FORMAT

GPR_$INQ_CONFIG (config, status)

INPUT PARAMETERS

none

OUTPUT PARAMETERS

config
Display configuration, in GPR_$DISPLAY_CONFIG_T format. The returned value for each display type is listed below:

Display Type	Returned Value
monochromatic portrait	GPR_$BW_800x1024
monochromatic landscape	GPR_$BW_1024x800.
color 1024 x 1024 (DN600) 2-board configuration	GPR_$COLOR_1024x1024x4
color 1024 x 1024, 3-borad configuration	GPR_$COLOR_1024x1024x8

status
Completion status, in STATUS_$T format.

GPR_$INQ_CONSTRAINTS — Returns the clipping window and plane mask used for the current bitmap.

FORMAT

```
GPR_$INQ_CONSTRAINTS (window, active, plane-mask, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

window
: The clipping window, in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, use a two-dimensional array.)

  The first element of GPR_$WINDOW_T specifies the window origin (top left corner coordinates), in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the bitmap limits.

  The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units. Width and height values, when added to the corresponding bitmap origin, must be within the bitmap limits.

active
: Boolean (logical) value which specifies whether the clip window is enabled. If the value is false, the clip window is disabled; if the value is true, the clip window is enabled.

plane-mask
: The plane mask, which specifies the active bitmap plane(s), in GPR_$MASK_T format. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* To establish a new clipping window for the current bitmap, use GPR_$SET_CLIP_WINDOW. To enable the new clipping window, use GPR_$SET_CLIPPING_ACTIVE. To establish a plane mask, use GPR_$SET_PLANE_MASK.

GPR_$INQ_COORDINATE_ORIGIN — Returns the x- and y-offsets added to all x- and y-coordinates used as input to move, line drawing, and BLT operations on the current bitmap.

FORMAT

```
GPR_$INQ_COORDINATE_ORIGIN (origin, status)
```

OUTPUT PARAMETERS

origin
: The current coordinate origin for the bitmap, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a new coordinate origin, use GPR_$SET_COORDINATE_ORIGIN.

GPR_$INQ_CP — Returns the current position in the current bitmap.

FORMAT

GPR_$INQ_CP (x, y, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

x
The x-coordinate of the current position, in GPR_$COORDINATE_T format. This is a 2-byte integer.

y
The y-coordinate of the current position, in GPR_$COORDINATE_T format. This is a 2-byte integer.

status
Completion status, in STATUS_$T format.

GPR_$INQ_CURSOR — Returns information about the cursor.

FORMAT

GPR_$INQ_CURSOR (curs-pat, curs-raster-op, active, position, origin, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

cursor-pat
: Identifier of the cursor pattern bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

cursor-raster-op
: Cursor raster operation code, in GPR_$RASTER_OP_ARRAY_T format. This is an eight-element array containing eight 2-byte integers. Currently, the value is 3. (The operation assigns all source values to the new destination).

active
: A Boolean (logical) value which indicates whether the cursor is displayed. The parameter is set to true if the cursor is displayed; it is set to false if the cursor is not displayed.

position
: The cursor's current position on the screen, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the cursor position's x-coordinate; the second element is the y-coordinate.

origin
: The pixel currently set as the cursor origin, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the pixel's cursor-relative x-coordinate; the second element is the pixel's cursor-relative y-coordinate.

status
: Completion status, in STATUS_$T format.

NOTES

* Cursor position: If a program calls this routine when in borrow-display mode, the x- and y-coordinates represent an absolute position on the screen. If a program calls this routine when the cursor is inside a frame of a display manager pad, the x- and y-coordinates are relative to the top left corner of the frame.

* To alter the cursor, use one of the following:
  GPR_$SET_CURSOR_PATTERN
  GPR_$SET_CURSOR_ACTIVE
  GPR_$SET_CURSOR_POSITION
  GPR_$SET_CURSOR_ORIGIN

* Currently, a program can not alter the cursor raster operation.

GPR_$INQ_DRAW_VALUE -- Returns the color/intensity value used for drawing lines.

FORMAT

```
GPR_$INQ_DRAW_VALUE (index, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

index
: The color map index that indicates the current color/intensity value used for drawing lines, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

  0-1 for monochromatic displays
  0-15 for color displays in 4-bit pixel format
  0-255 for color displays in 8-bit or 24-bit pixel format
  -1 for all displays. This specifies that the background is transparent; that is, the old values of the pixels are not changed.
  -2 for all displays. This specifies using the color/intensity value of the bitmap background as the line drawing value. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a new draw value, use GPR_$SET_DRAW_VALUE.

GPR_$INQ_FILL_BACKGROUND_VALUE — Returns the color/intensity value of the background used for tile fills.

FORMAT

```
GPR_$INQ_FILL_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

index
: The color map index that indicates the current color/intensity value used for tile fills, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

  0-1 for monochromatic displays
  0-15 for color displays in 4-bit pixel format
  0-255 for color displays in 8-bit or 24-bit pixel format
  -1 for all displays. This specifies that the background is transparent; that is, the old values of the pixels are not changed.
  -2 for all displays. This specifies using the color/intensity value of the bitmap background as the tile fill background. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a new background value, use GPR_$SET_FILL_BACKGROUND_VALUE.

GPR_$INQ_FILL_PATTERN — Returns the fill pattern for the current bitmap.

FORMAT

```
GPR_$INQ_FILL_PATTERN (pattern, scale, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

pattern
: The descriptor of the bitmap containing the fill pattern, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

scale
: The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a new fill pattern for the current bitmap, use GPR_$SET_FILL_PATTERN.

* Currently, the tile pattern must be stored in a bitmap that is 32x32 pixels by n planes. The scale factor must be one. Any other pattern size or scale value results in an error.

* With a one-plane bitmap as the pattern, the pixel values used are those set by GPR_$SET_FILL_VALUE and GPR_$SET_FILL_BACKGROUND_VALUE. Pixels corresponding to "1" bits of the pattern are drawn in the fill value: pixels corresponding to "0" bits of the pattern are drawn in the fill background value.

GPR_$INQ_FILL_VALUE -- Returns the color/intensity value used to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
GPR_$INQ_FILL_VALUE (index, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

index
: The color map index that indicates the current color/intensity fill value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

    0-1 for monochromatic displays
    0-15 for color displays in 4-bit pixel format
    0-255 for color displays in 8-bit or 24-bit pixel format

status
: Completion status, in STATUS_$T format.

NOTES

* To set a new fill value, use GPR_$SET_FILL_VALUE.

GPR_$INQ_HORIZONTAL_SPACING -- Returns the parameter for the width of spacing between displayed characters for the specified font.

FORMAT

```
GPR_$INQ_HORIZONTAL_SPACING (font-id, horizontal-spacing, status)
```

INPUT PARAMETERS

font-id
: Identifier of the text font. This is a 2-byte integer.

OUTPUT PARAMETERS

horizontal-spacing
: The parameter for horizontal spacing of the specified font. This is a 2-byte integer. Possible values are -127 to 127.

status
: Completion status, in STATUS_$T format.

NOTES

* Use GPR_$SET_HORIZONTAL_SPACING to set the width of spacing for a font.
* The initial width of horizontal spacing is defined in the font file.
* This routine returns the horizontal spacing in the local copy of the font. Initially, this is a copy of the font file; however, the local copy may have been changed. Change in the local copy does not affect the font file or the use of the font by other processes.

GPR_$INQ_IMAGING_FORMAT — Returns the current imaging format.

FORMAT

GPR_$INQ_IMAGING_FORMAT (format, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

format
Imaging format in GPR_$IMAGING_FORMAT_T configuration. If you are using an interactive format, the returned value is GPR_$INTERACTIVE.

If you are using the imaging 8-bit pixel format on a two-board configuration, the returned value is GPR_$IMAGING_1024x1024x8. If you are using the imaging 24-bit pixel format, the returned value is GPR_$IMAGING_512x512x24.

status
Completion status, in STATUS_$T format.

NOTES

* See Section 2.3 for more information about imaging formats.

GPR_$INQ_LINE_PATTERN — Returns the pattern used in drawing lines.

FORMAT

GPR_$INQ_LINE_PATTERN (repeat, pattern, length, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

repeat
: The replication factor for each bit in the pattern. This is a 2-byte integer.

pattern
: The bit pattern, left justified, in GPR_$LINE_PATTERN_T format. This is a four-element array of 2-byte integers.

length
: The length of the pattern in bits. This is a 2-byte integer in the range of 0 to 64.

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$INQ_LINE_PATTERN returns the current line pattern set explicitly with GPR_$SET_LINE_PATTERN or set implicitly with GPR_$SET_LINESTYLE.

* Use GPR_$SET_LINE_PATTERN to specify a new line pattern. You can also use GPR_$SET_LINESTYLE to set a line pattern within the limits of the parameter GRP_$DOTTED.

GPR_$INQ_LINESTYLE — Returns information about the current line-style.

FORMAT

GPR_$INQ_LINESTYLE (style, scale, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

style
: The style of line, in GPR_$LINESTYLE_T format. Possible values are GPR_$SOLID and GPR_$DOTTED. This is a 2-byte integer.

scale
: The scale factor for dashes if the style parameter is GPR_$DOTTED. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* When the line-style attribute is GPR_$DOTTED, lines are drawn in dashes. The scale factor determines the number of pixels in each dash and in each space between the dashes.

* To set the line-style attribute, use GPR_$SET_LINESTYLE.

GPR_$INQ_RASTER_OPS -- Returns the raster operations for the current bitmap.

FORMAT

GPR_$INQ_RASTER_OPS (raster-op, status)

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

raster-op
: Raster operation codes, in GPR_$RASTER_OP_ARRAY_T format. This is an eight-element array of 2-byte integers. Each element corresponds to the raster operation for a single plane of the bitmap. Possible raster op values are zero through fifteen.

status
: Completion status, in STATUS_$T format.

NOTES

* See Table 3-1 for a listing of the raster operation codes and their functions.
* To set a new raster operation for the current bitmap, use GPR_$SET_RASTER_OP.

GPR_$INQ_SPACE_SIZE — Returns the width of the space to be displayed when a character requested is not in the specified font.

FORMAT

```
GPR_$INQ_SPACE_SIZE (font-id, space-size, status)
```

INPUT PARAMETERS

font-id
: Identifier of the text font. This is a 2-byte integer.

OUTPUT PARAMETERS

space-size
: The space size of the specified font. This is a 2-byte integer. Possible values are -127 to 127.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a font's space size, use GPR_$SET_SPACE_SIZE.
* The initial space size is defined in the font file.
* The space size is the number of pixels to skip in the horizontal direction when a character that is not in the font is written.

GPR_$INQ_TEXT -- Returns the text font and text path used for the current bitmap.

FORMAT

```
GPR_$INQ_TEXT (font-id, path, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

font-id
: Identifier of the text font used for the current bitmap. This is a 2-byte integer.

path
: The direction of movement from one text character position to the next in the current bitmap, in GPR_$DIRECTION_T format. Possible values are GPR_$UP, GPR_$DOWN, GPR_$LEFT, and GPR_$RIGHT.

status
: Completion status, in STATUS_$T format.

NOTES

* To set a new text font for the current bitmap, use GPR_$SET_TEXT_FONT.

GPR_$INQ_TEXT_EXTENT — Returns the x- and y-offsets a string spans when written by GPR_$TEXT.

FORMAT

GPR_$INQ_TEXT_EXTENT (string, string-length, size, status)

INPUT PARAMETERS

string
: A string, in GPR_$STRING_T format. This is an array of characters.

string-length
: Number of characters in the string. This is a 2-byte integer. The maximum value is 256.

OUTPUT PARAMETERS

size
: Width and height of the area the written string would occupy, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the width (the x-offset); the second element is the height (the y-offset).

status
: Completion status, in STATUS_$T format.

NOTES

* When the text path is GPR_$RIGHT or GPR_$LEFT, the width is the x-offset. When the text path is GPR_$UP or GPR_$DOWN, the height is the y-offset.

* See GPR_$SET_TEXT_PATH for use of GPR_$RIGHT, GPR_$LEFT, GPR_$UP, and GPR_$DOWN.

* Figure 11-1 shows two examples of the extent of text in relation to offsets. For horizontal text, use GPR_$RIGHT with GPR_$SET_TEXT_PATH. For rotated text, use GPR_$UP with GPR_$SET_TEXT_PATH.

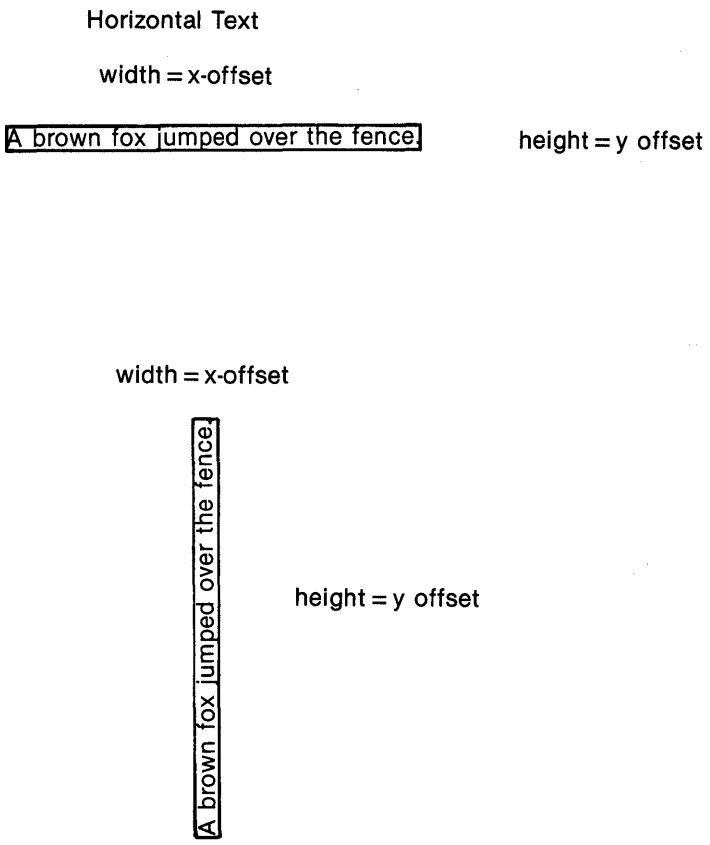


Figure 11-1. Height and Width for Horizontal and Rotated Text

GPR_$INQ_TEXT_OFFSET — Returns the x- and y-offsets from the top left pixel of a string to to the origin of the string's first character. This routine also returns the x- or y-offset to the pixel which is the new current position after the text is written with GPR_$TEXT.

FORMAT

```
GPR_$INQ_TEXT_OFFSET (string, string-length, start, xy-end, status)
```

INPUT PARAMETERS

string
A string, in GPR_$STRING_T format. This is an array of characters.

string-length
Number of characters in the string. This is a 2-byte integer. The maximum value is 256.

OUTPUT PARAMETERS

start
X- and y-offsets from the top left pixel of the string to the origin of its first character, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the x-offset; the second element is the y-offset.

xy-end
The X- or Y-offset from the top left pixel of the string to the pixel that will be the new current position after the string is written with GPR_$TEXT. This is the X-offset when the text path is specified as GPR_$RIGHT or GPR_$LEFT. This is the Y-offset when the text path is specified as GPR_$UP or GPR_$DOWN. This is a 2-byte integer.

status
Completion status, in STATUS_$T format.

NOTES

* A program can use the information derived from the "start" output parameter to set the current position to the character origin, rather than the top left corner of the string, before writing the string with GPR_$TEXT.

* When the text path is GPR_$RIGHT or GPR_$LEFT, the offset is to the x-axis. When the text path is GPR_$UP or GPR_$DOWN, the offset is to the y-axis.

* See GPR_$SET_TEXT_PATH for use of GPR_$RIGHT, GPR_$LEFT, GPR_$UP, and GPR_$DOWN.

* Figure 11-2 shows an example of text offsets, after using GPR_$RIGHT and GPR_$UP with GPR_$SET_TEXT_PATH.

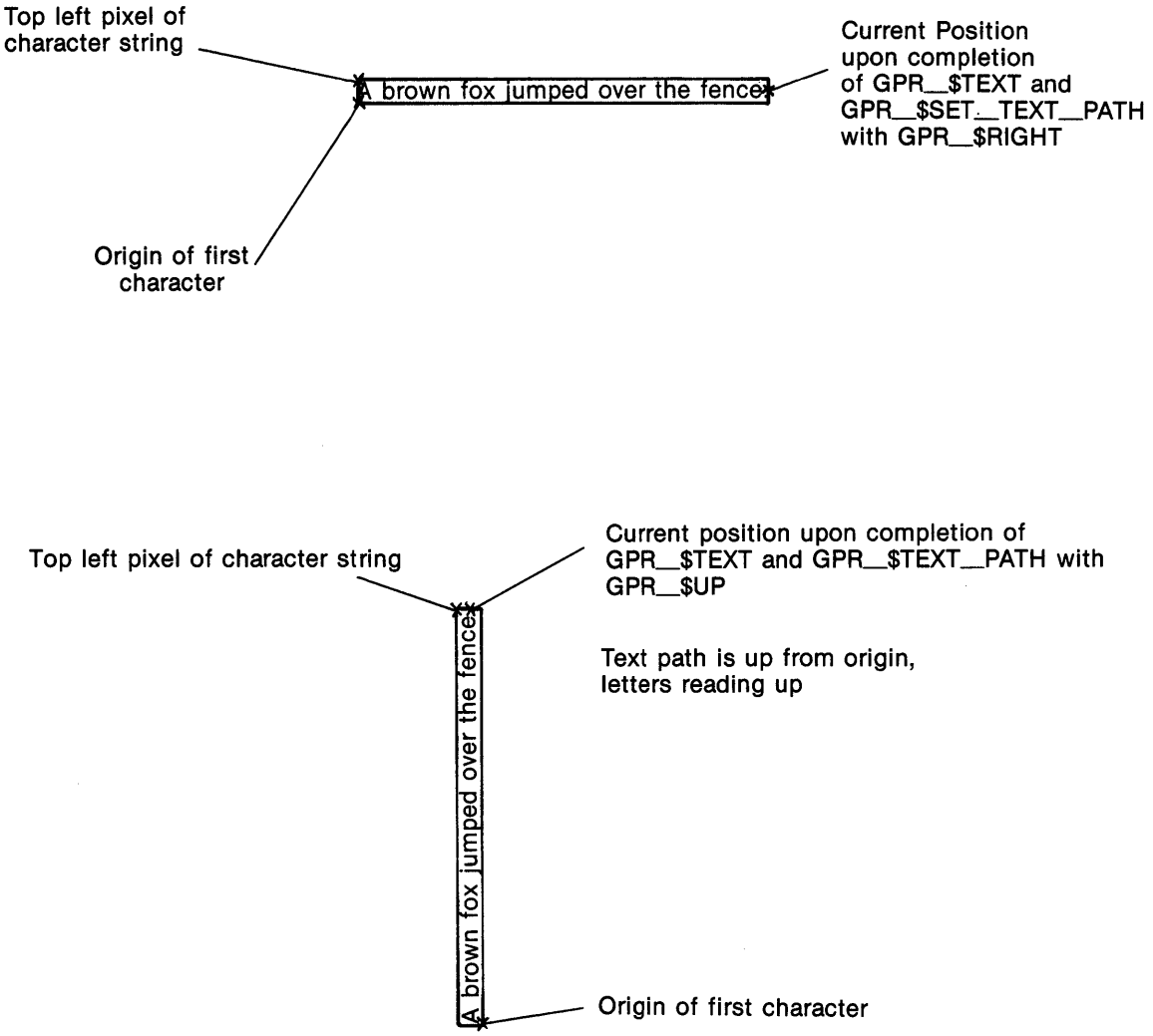


Figure 11-2. Text Offsets

GPR_$INQ_TEXT_PATH — Returns the direction for writing a line of text.

FORMAT

```
GPR_$INQ_TEXT_PATH (direction, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

direction
: Direction for writing text, in GPR_$DIRECTION_T format. Possible values are GPR_$UP, GPR_$DOWN, GPR_$LEFT, and GPR_$RIGHT.

status
: Completion status, in STATUS_$T format.

NOTES

* To set the current text path, use GPR_$SET_TEXT_PATH.

GPR_$INQ_TEXT_VALUES — Returns the current values of text and text background color/intensity value used in the current bitmap.

FORMAT

```
GPR_$INQ_TEXT_VALUES (text-value, text-bkgd-value, status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

text-value
: A color map index that indicates the text color/intensity value, in GPR_$PIXEL_VALUE_T format.

text-bkgd-value
: A color map index that indicates the text background color/intensity value, in GPR_$PIXEL_T format.

status
: Completion status, in STATUS_$T format.

NOTES

* To establish the text color/intensity value, use GPR_$SET_TEXT_VALUE. To establish the text background color/intensity value, use GPR_$SET_TEXT_BACKGROUND_VALUE.

GPR_$INQ_VIS_LIST — Returns a list of the visible sections of an obscured window.

FORMAT

GPR_$INQ_VIS_LIST (slots-available, slots-total, vis-list, status)

INPUT PARAMETERS

slots-available
: The size of the array of visible window sections. This is a 2-byte integer, which is the maximum number of visible rectangles that can be returned. If you want to list all existing sections, you must specify a number that is greater than or equal to the number returned in the slots-total argument.

OUTPUT PARAMETERS

slots-total
: The number of existing visible rectangles. This is a 2-byte integer. If this value is greater than the slots-available parameter, then only the number of rectangles specified in slots-available is returned.

vis-list
: The list of visible window sections. This is an array in GPR_$WINDOW_T format. GPR_$WINDOW_T format consists of a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

  The first element of GPR_$WINDOW_T specifies the window section's base relative to the top left corner of the window, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the x-coordinate of the base; the second is the y-coordinate.

  The second element of GPR_$WINDOW_T specifies the section's size, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the section width in raster units; the second element is the section height.

  There is no set limit to the number of visible regions that may be returned.

status
: Completion status, in STATUS_$T format.

NOTES

* If the display has been acquired but the target window is obscured, programs can call GPR_$INQ_VIS_LIST to locate any visible sections of the window.

* If the target window is visible, this routine returns a base of (0,0) and the size of the entire window.

* If the window is obscured, the application should call GPR_$SET_CLIP_WINDOW once for each rectangle returned by GPR_$INQ_VIS_LIST before making calls to drawing routines. Clipping is to rectangles only. The GPR software will not perform clipping automatically.

* GPR_$INQ_VIS_LIST implicitly releases and reacquires the display in order to communicate with the Display Manager.

GPR_$INQ_WINDOW_ID — Returns the character that identifies the current bitmap's window.

FORMAT

```
GPR_$INQ_WINDOW_ID (character, status)
```

INPUT PARAMETERS

none

OUTPUT PARAMETERS

character
: The character that identifies the current bitmap's window.

status
: Completion status, in STATUS_$T format.

NOTES

* This character is returned by GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT when they return GPR_$ENTERED_WINDOW events. The character indicates which window was entered.

* The character 'A' is the default value of the window identification for all windows.

GPR_$LINE — Draws a line from the current position to the given position, then updates the current position to the given position.

FORMAT

```
GPR_$LINE (x,y, status)
```

INPUT PARAMETERS

x
: The x-coordinate, which designates the end point of the line and then becomes the current x-coordinate. Use GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within the bitmap limits, unless clipping is enabled.

y
: The y-coordinate, which designates the end point of the line and then becomes the current y-coordinate. Use GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within the bitmap limits, unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the line drawn.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

* After the line has been drawn, its end point becomes the current position.

* To set a new position without drawing a line, use GPR_$MOVE.

GPR_$LOAD_FONT_FILE — Loads a font from a file into the display's font storage area.

FORMAT

```
GPR_$LOAD_FONT_FILE (pathname, pathname-length, font-id, status)
```

INPUT PARAMETERS

pathname
: Pathname of the file containing the font, in NAME_$PNAME_T format. This is a character string.

pathname-length
: Number of characters in font file pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

font-id
: Font identifier. This is a 2-byte integer.

status
: Completion status, in STATUS_$T format.

NOTES

* Use the font-id returned from this file as input for GPR_$SET_TEXT_FONT.
* To unload fonts loaded with this routine, use GPR_$UNLOAD_FONT_FILE.

GPR_$MOVE — Changes the current position.

FORMAT

GPR_$MOVE (x, y, status)

INPUT PARAMETERS

x
The x-coordinate, which becomes the current x-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

y
The y-coordinate, which becomes the current y-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* The current position is the starting point for any line drawing and text operations.

* GPR_$MOVE does not draw any lines.

* See Section 6.1 for more information about the current position and the move routine.

* The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the move operation.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$MULTILINE — Draws a series of disconnected lines.

FORMAT

GPR_$MULTILINE (x, y, npositions, status)

INPUT PARAMETERS

x
List of the x-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits, unless clipping is enabled.

y
List of the y-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array The values must be within the bitmap limits, unless clipping is enabled.

npositions
Number of coordinate positions. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$MULTILINE alternately moves to a new position and draws lines: it moves to the first given position, draws a line from the first to the second given position, moves to the third position, etc. After the last line has been drawn or the last move has been made, the endpoint becomes the current position.

* The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the multiline drawn.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$MULTITRAPEZOID -- Draws and fills a list of trapezoids in the current bitmap.

FORMAT

GPR_$MULTITRAPEZOID (trapezoid-list, trapezoid-number, status)

INPUT PARAMETERS

trapezoid-list
The trapezoids to fill. This is an array of one or more trapezoids in GPR_$TRAP_T format, a two-element array in GPR_$HORIZ_SEG_T format. The first element is the top horizontal segment of the trapezoid; the second element is the bottom horizontal segment.

trapezoid-number
The number of trapezoids to fill. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$MULTITRAPEZOID fills in a list of trapezoids with the color/intensity value specified with GPR_$SET_FILL_VALUE.

* To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

GPR_$OPEN_BITMAP_FILE — Opens a file for external storage of a bitmap.

FORMAT

```
GPR_$OPEN_BITMAP_FILE (access, filename, name-size, version, size, groups,
                       group-headers, attributes, bitmap, created, status)
```

INPUT PARAMETERS

access
One of four ways to access external bitmap objects, in GPR_$ACCESS_MODE_T format. Possible values for this parameter are:

GPR_$CREATE
GPR_$UPDATE
GPR_$WRITE
GPR_$READONLY

filename
The pathname of the bitmap file, in NAME_$PNAME_T format.

name-size
The length of the file name. This is a 2-byte integer.

version
The version number on the header of the external bitmap file, in GPR_$VERSION_T format. This is a two-element array of two 2-byte integers: a major version number and a minor version number. Currently, version is not used and is always returned as major version 1, minor version 1.

size
Bitmap width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is bitmap width, in raster units; the second element is the bitmap height, in raster units. Possible values for x are 1-4096; possible values for y are 1-4096.

groups
The number of groups in external bitmaps. This is a 2-byte integer. Possible values are 1..(GPR_$MAX_BMF_GROUP +1). Currently, groups are not used; this value must be 1.

group-headers
Descriptors of the external bitmap group headers, in GPR_$BMF_GROUP_HEADER_ARRAY_T format. This is an array [0..GPR_$MAX_BMF_GROUP] of GPR_$BMF_GROUP_HEADER_T. Possible values for fields in a group header are the following. Alignment constraints are included:

N_SECTS	2-byte integer, 1..8
PIXEL_SIZE	2-byte integer Currently, this value must be 1.
ALLOCATED_SIZE	2-byte integer Currently, this value must be 1.
BYTES_PER_LINE	2-byte integer This value must be a multiple of 4 This value is usually specified as 0.
BYTES_PER_SECT	4-byte integer This value must be either rounded up to a page boundary, or for small bitmaps rounded up to the next largest binary submultiple of a page, for example, one-half, one-fourth, or one-eighth. One page equals 1024 bytes. This value is usually specified as 0.
STORAGE_OFFSET	UNIV_PTR format

attribs
: The attributes which the bitmap will use, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

bitmap
: Descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

created
: Boolean (logical) value which specifies whether the bitmap file was created. If the value is true, the file was created.

status
: Completion status, in STATUS_$T format.

NOTES

* See Chapter 10 for sample programs that use GPR_$OPEN_BITMAP_FILE.

* Currently, a section is equivalent to a bit plane. N_SECTS may include up to eight bit planes.

* For ALLOCATED_SIZE, BYTES_PER_LINE and BYTES_PER_SECT, you can specify values as 0, and the GPR package will calculate and return the appropriate values.

* BYTES_PER_SECT is not necessarily a multiple of BYTES_PER_LINE. This means that GPR will leave unused space at the end of one section to satisfy alignment constraints. The result is that the next section starts on an alignment boundary, which is normally a page boundary.

* The access parameter specifies one of four ways to use external bitmaps. As shown in the table below, the value given for this parameter determines whether four other parameters are input (IN) or output (OUT). The values for these parameters are used to validate your input with GPR_$CREATE and GPR_$UPDATE.

	GPR_$CREATE	GPR_$UPDATE file exists no	GPR_$UPDATE file exists yes	GPR_$WRITE	GPR_$READONLY
version, size, groups, group-headers	IN	IN	OUT	OUT	OUT

* GPR_$CREATE indicates that you want a new external bitmap file. GPR_$UPDATE means that you want to create a new file or overwrite an existing one.

* When you specify GPR_CREATE as the access parameter and you specify a file name that already exists, the file is superseded only if it is a bitmap file. If the file is not a bitmap file, you get the error message NAME_$ALREADY_EXISTS.

* Attributes are not stored with the bitmap. You assign attributes when you open the bitmap file. See the routines GPR_$ALLOCATE_ATTRIBUTE_BLOCK and GPR_$ALLOCATE_BITMAP.

* FORTRAN programmers can use the following equivalences:

```
Parameter           (n_groups =1)

Integer*2           headers2(8,n_groups)
Integer*4           headers4(4,n_groups)

Equivalence         headers2(1,1),headers4(1,1)

  N_SECTS           headers2(1,n)
  PIXEL_SIZE        headers2(2,n)
  ALLOCATED_SIZE    headers2(3,n)
  BYTES_PER_LINE    headers2(4,n)
  BYTES_PER_SECT    headers4(3,n)
  STORAGE_OFFSET    headers4(4,n)
```

* Figure 11-3 is a global view of one group.

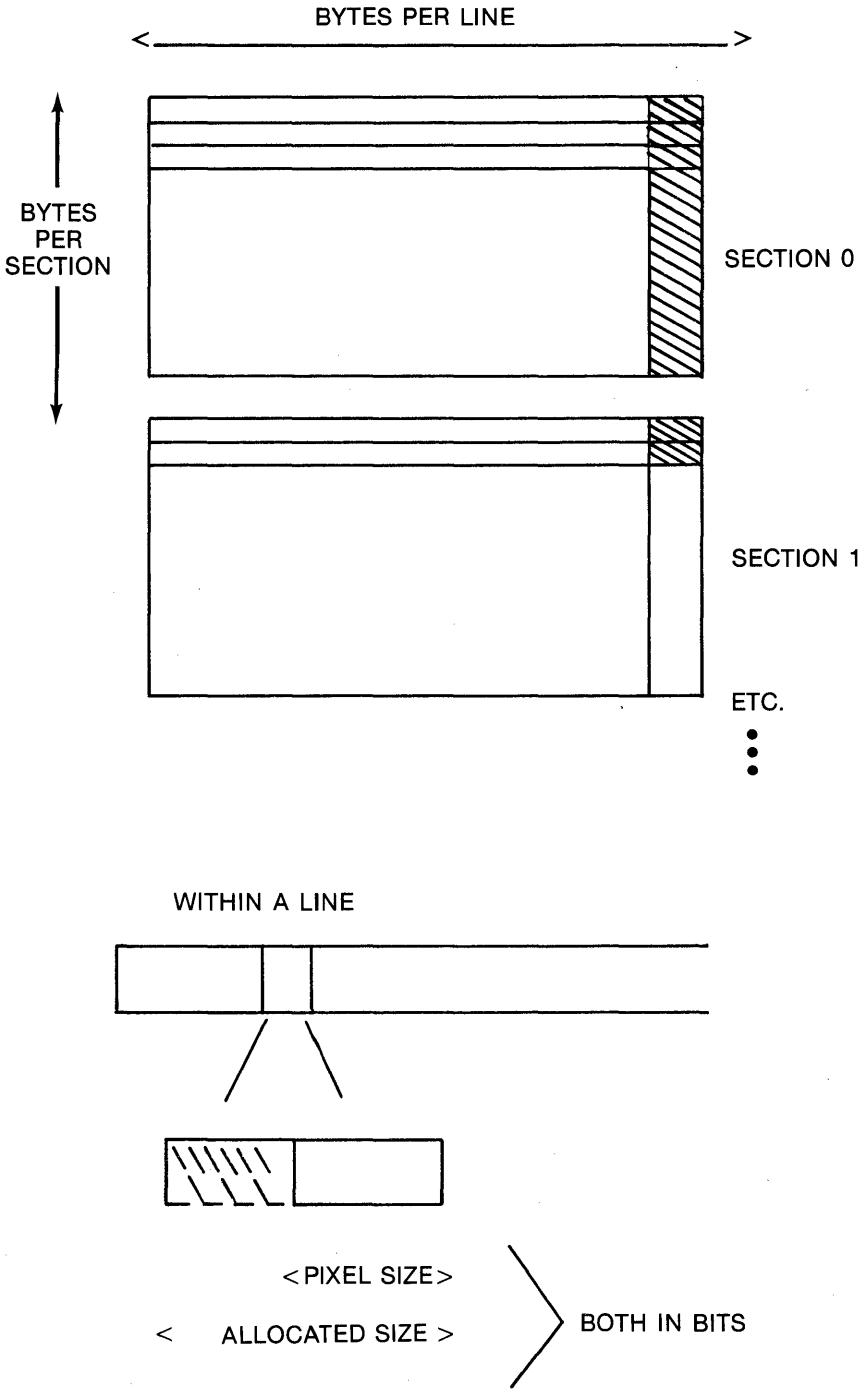


Figure 11-3. View of One Group

GPR_$PGON_POLYLINE -- Defines a series of line segments forming part of a polygon boundary.

FORMAT

GPR_$PGON_POLYLINE (x, y, npositions, status)

INPUT PARAMETERS

x
: List of the x-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits, unless clipping is enabled.

npositions
: Number of coordinate positions. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$PGON_POLYLINE defines a series of line segments that comprise part of a polygon to be filled in by either (1) GPR_$CLOSE_FILL_PGON or by (2) GPR_$CLOSE_RETURN_PGON and GPR_$MULTITRAPEZOID. The lines are not drawn on the screen until the polygon is filled in by either routines (1) or (2) above. To draw a series of connected lines without filling the resulting figure, use GPR_$POLYLINE.

* GPR_$PGON_POLYLINE must be called only when the line segments of a polygon are being defined; see the routine GPR_$START_PGON for more information.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$PIXEL_BLT — Performs a pixel block transfer from any bitmap to the current bitmap.

FORMAT

GPR_$PIXEL_BLT (source-bitmap-desc, source-window, dest-origin, status)

INPUT PARAMETERS

source-bitmap-desc
: Descriptor of the source bitmap which contains the source window to be transferred, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

source-window
: Rectangular section of the bitmap from which to transfer pixels, in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

  The first element of GPR_$WINDOW_T specifies the origin of the window (top-left corner), in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the source bitmap.

  The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of two-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units.

dest-origin
: Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the current bitmap, unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Use GPR_$SET_BITMAP to establish the current bitmap for this routine.

* Both the source and destination bitmaps can be in either display memory or main memory.

* The source window origin is added to the coordinate origin for the source bitmap, and the result is the actual origin of the source rectangle for the BLT. Similarly, the destination origin is added to the coordinate origin for the current bitmap, and the result is the actual origin of the destination rectangle for the BLT.

LIMITATIONS

* If the source bitmap is a Display Manager frame, the only allowed raster op codes are 0, 5, A, and F. These are the raster operations in which the source plays no role.

* If a rectangle is transferred by a BLT to a Display Manager frame and the frame is refreshed for any reason, the BLT is re-executed. Therefore, if the information in the source bitmap has changed, the appearance of the frame changes accordingly.

GPR_$POLYLINE — Draws a series of connected lines.

FORMAT

GPR_$POLYLINE (x, y, npositions, status)

INPUT PARAMETERS

x
List of the x-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits, unless clipping is enabled.

y
List of the y-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits, unless clipping is enabled.

npositions
Number of coordinate positions. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$POLYLINE draws a series of connected lines: it starts from the current position, draws to the first given position, then sets the current position to the first given position. This is repeated for all given positions.

* The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the polyline drawn.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

* See Section 6.2 for more information about drawing polylines.

GPR_$READ_PIXELS — Reads the pixel values from a window of the current bitmap and stores the values in a pixel array.

FORMAT

GPR_$READ_PIXELS (source-window, pixel-array, status)

INPUT PARAMETERS

source-window
Rectangular section of the current bitmap from which to read pixel values (color/intensity), in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

The first element of GPR_$WINDOW_T specifies the window origin (the top left corner coordinates), in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within bitmap limits.

The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units. Width and height values, when added to the window origin, must be within bitmap limits.

OUTPUT PARAMETERS

pixel-array
Array of the pixel values (color/intensity). This is an array of 4-byte integers. GPR_$PIXEL_ARRAY_T format is an example of such an array.

status
Completion status, in STATUS_$T format.

NOTES

* The pixel values from the source window of the current bitmap are stored in the pixel array in row-major order, one in each 4-byte integer.

* To write pixel values from an array to the current bitmap, use GPR_$WRITE_PIXELS.

LIMITATIONS

* A program cannot use this routine on a bitmap corresponding to a Display Manager frame.

* A program cannot read pixels values in imaging formats.

GPR_$RECTANGLE -- Fills a rectangle.

FORMAT

GPR_$RECTANGLE (rectangle, status)

INPUT PARAMETERS

rectangle
The rectangle in the current bitmap to be filled in. Rectangle is in GPR_$WINDOW_T format. This is a two-element array of two-element arrays.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$RECTANGLE fills in a rectangle with the color/intensity value specified with GPR_$SET_FILL_VALUE. To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

* See Section 6.2 for more information about filling rectangles.

GPR_$RELEASE_DISPLAY -- Decrements a counter associated with the number of times a display has been acquired.

FORMAT

```
GPR_$RELEASE_DISPLAY (status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

status
    Completion status, in STATUS_$T format.

NOTES

* GPR_$RELEASE_DISPLAY decrements a counter whose value reflects the number of times the display has been acquired. If the counter value reaches zero, the routine releases the display, allowing other processes, including the Display Manager, to use the display.

* GPR_$RELEASE_DISPLAY automatically releases the keyboard when it releases the display.

* Programs that call GPR_$EVENT_WAIT may not need to call GPR_$RELEASE_DISPLAY, since GPR_$EVENT_WAIT releases the display implicitly whenever the process waits for input.

GPR_$REMAP_COLOR_MEMORY — Establishes the single plane of color display memory to be accessed directly.

FORMAT

GPR_$REMAP_COLOR_MEMORY (plane, status)

INPUT PARAMETERS

plane
: Plane for access through storage-pointer, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are 0-7.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* When accessing color display memory directly (i.e. by dereferencing the pointer returned by GPR_$INQ_BITMAP_POINTER), the program can access only one plane at a time. (This is unlike access to multi-plane memory bitmaps, in which the first scan line of a plane immediately follows the last scan line of the previous plane in virtual memory.) Therefore, a program must use GPR_$REMAP_COLOR_MEMORY to establish which plane of color display memory will be accessible through the pointer returned by GPR_$INQ_BITMAP_POINTER.

GPR_$REMAP_COLOR_MEMORY_1 — Sets the plane of hidden color display memory mapped at the address returned by GPR_$INQ_BITMAP_POINTER.

FORMAT

GPR_$REMAP_COLOR_MEMORY_1 (plane, status)

INPUT PARAMETERS

plane
Plane for access through storage-pointer, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are 0-7.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$REMAP_COLOR_MEMORY_1 makes visible the normally hidden frame 1 of color display memory. GPR_$REMAP_COLOR_MEMORY sets frame 0.

GPR_$REPLICATE_FONT -- Creates and loads a modifiable copy of a font.

FORMAT

GPR_$REPLICATE_FONT (font-id, repli-font-id, status)

INPUT PARAMETERS

font-id
    Identifier of the text font. This is a 2-byte integer.

OUTPUT PARAMETERS

repl-font-id
    Identifier of the copied text font. This is a 2-byte integer.

status
    Completion status, in STATUS_$T format.

NOTES

* For a description of fonts and font files, see the DOMAIN System Command Reference Manual.

* To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

GPR_$SELECT_COLOR_FRAME — Selects whether frame 0 or frame 1 of a plane of color display memory is visible.

FORMAT

```
GPR_$SELECT_COLOR_FRAME (frame, status)
```

INPUT PARAMETERS

frame
: A 2-byte integer that denotes which frame is to be visible. Possible values are zero or one.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Normally, frame 0 is visible.

GPR_$SET_ACQ_TIME_OUT — Establishes the length of time the display will be acquired.

FORMAT

```
GPR_$SET_ACQ_TIME_OUT (timeout, status)
```

INPUT PARAMETERS

timeout
: The maximum real time, in TIME_$CLOCK_T format, for which the program can acquire the display.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* If the program has not released the display when the time-out expires and another process (for example, the Display Manager) needs the display, an acquire time-out fault (SMD_$ACQUIRE_TIMEOUT) is generated in the user process. The acquire time-out fault is a warning fault that the program can intercept with a cleanup handler or static fault handler. If the program does not release the display within a few seconds of the acquire timeout fault, a second fault occurs (with the status code FAULT_$QUIT) and the program loses control of the display.

* If this routine is not called, the default time-out value is one minute.

GPR_$SET_ATTRIBUTE_BLOCK — Associates an attribute block with the current bitmap.

FORMAT

GPR_$SET_ATTRIBUTE_BLOCK (attrib-block-desc, status)

INPUT PARAMETERS

attrib-block-desc
The descriptor of the attribute block, in GPR_$ATTRIBUTE_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To allocate and deallocate attribute blocks, use GPR_$ALLOCATE_ATTRIBUTE_BLOCK and GPR_$DEALLOCATE_ATTRIBUTE_BLOCK.

* To request the descriptor of the current bitmap's attribute block, use GPR_$ATTRIBUTE_BLOCK.

GPR_$SET_AUTO_REFRESH -- Directs the Display Manager to refresh the window automatically.

FORMAT

```
GPR_$SET_AUTO_REFRESH (auto-refresh, status)
```

INPUT PARAMETERS

auto-refresh
: A Boolean value that indicates whether or not the Display Manager will automatically refresh the application's window. A value of true means that auto-refresh is enabled; a value of false (the default) means that auto-refresh is disabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Automatic refresh of windows can affect system performance and reduce the amount of disk space available, especially if the application's windows are large.

* As an alternative, the application program can also provide procedures that refresh the screen and hidden display; see the routine GPR_$SET_REFRESH_ENTRY.

* This routine implicitly releases and reacquires the display in order to communicate with the Display Manager.

* This routine applies to the current bitmap. When a program changes attribute blocks for a bitmap during a graphics session, the auto refresh flag is lost unless you set it for the new attribute block.

GPR_$SET_BITMAP — Establishes a bitmap as the current bitmap for subsequent operations.

FORMAT

```
GPR_$SET_BITMAP (bitmap-desc, status)
```

INPUT PARAMETERS

bitmap-desc
: The bitmap descriptor, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer that uniquely identifies the bitmap.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The program can obtain the bitmap descriptor by using GPR_$INQ_BITMAP.
* After a bitmap is established, it is called the "current bitmap".

GPR_$SET_BITMAP_DIMENSIONS -- Modifies the size and the number of planes of a bitmap.

FORMAT

GPR_$SET_BITMAP_DIMENSIONS (bitmap-desc, size, hi-plane-id, status)

INPUT PARAMETERS

bitmap-desc
: The descriptor of the bitmap, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

size
: New width and height of the bitmap, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the bitmap width, in raster units; the second element is the bitmap height, in raster units.

hi-plane-id
: The new identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* A program can use this call to change the size of a bitmap after the bitmap has been created. This is useful if the program wishes to restrict itself to an upper-left subset of the original bitmap, or to use hidden memory on a borrowed display.

* In direct mode, when you allocate a bitmap, you request a size. You may get a smaller size if the Display Manager window is smaller than the size you requested. These restrictions apply to resizing bitmaps. Any bitmap can be shrunk from its original dimensions in x, y or the highest plane. Once the bitmap has been shrunk, it can grow up to its requested size. The maximum allowed sizes for x, y and the highest plane for the various DOMAIN displays are given in the following table.

	max X	max Y	max high plane
Monochromatic display (either portrait or landscape)	1024	1024	0
Color display—Interactive format			
4-bit pixels	1024	2048	3
8-bit pixels	1024	2048	7

* If a program uses hidden display memory, it must be careful not to modify areas that are being used to store fill constants or text fonts. The following areas may be used by these functions on the various DOMAIN displays.

Fill constants:
Both monochromatic displays: 800 <= X <= 1023 and Y = 1023.
Color displays: none.

Stand-alone font:
Monochromatic portrait display: 800 <= X <= 1023 and 0 <= Y <= 39.
Monochromatic landscape display: 800 <= X <= 1023 and 983 <= Y <= 1022.
Color displays: same as monochromatic portrait display, plane 0 only, Y offset by 1024.

User text fonts: (only if text fonts are loaded)
Monochromatic portrait display: 800 <= X <= 1023 and 40 <= Y <= 1022, allocated from top to bottom.
Monochromatic landscape display: 0 <= X <= 1023 and 800 <= Y <= 1023, in columns 224 bits wide, allocated top to bottom and left to right.
Color displays: same as monochromatic portrait display, plane 0 only, Y offset by 1024.

Note that these areas may move, grow or shrink in future DOMAIN software releases. Therefore, only limited use should be made of hidden display memory in conjunction with text or cursor operations.

GPR_$SET_CHARACTER_WIDTH — Specifies the width of the specified character in the specified font.

FORMAT

```
GPR_$SET_CHARACTER_WIDTH (font-id, character, width, status)
```

INPUT PARAMETERS

font-id
: Identifier of the text font. This is a 2-byte integer.

character
: The specified character. This is a CHAR variable.

width
: The width parameter of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* To retrieve a character's width, use GPR_$INQ_CHARACTER_WIDTH.
* The initial character widths are defined in the font file. For a description of fonts and font files, see the DOMAIN System Command Reference Manual.
* To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

GPR_$SET_CLIPPING_ACTIVE — Enables/disables a clipping window for the current bitmap.

FORMAT

```
GPR_$SET_CLIPPING_ACTIVE (active, status)
```

INPUT PARAMETERS

active
: A Boolean (logical) value which specifies whether or not to enable the clipping window. Set this value to true to enable the clipping window; set it to false to disable the clipping window.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* To specify a clipping window, use the routine GPR_$SET_CLIP_WINDOW.
* Initially, in borrow-display and frame modes, the clip window is disabled. In direct mode, the clip window is enabled and clipped to the size of the window.
* To inquire whether the clip window is enabled, use GPR_$INQ_CONSTRAINTS.

GPR_$SET_CLIP_WINDOW -- Changes the clipping window for the current bitmap.

FORMAT

GPR_$SET_CLIP_WINDOW (window, status)

INPUT PARAMETERS

window
The new clipping window, in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

The first element of GPR_$WINDOW_T specifies the window origin (the top left corner coordinate position) in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the bitmap limits.

The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units. Width and height values, when added to the corresponding origin coordinate, must be within bitmap limits. (See Figure 11-4.)

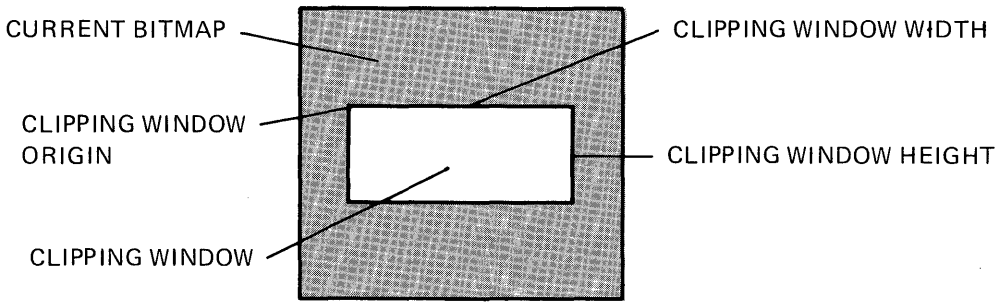


Figure 11-4. Clipping Window Origin, Width, Height

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* The default clip window is the entire bitmap.
* Pixels outside the clip window in the current bitmap are not modified by subsequent operations.
* To enable the clip window, use GPR_$SET_CLIPPING_ACTIVE.
* To request the dimensions of the current clip window, use GPR_$INQ_CONSTRAINTS.

LIMITATIONS

* This call is not allowed on the bitmap corresponding to the Display Manager frame.

GPR_$SET_COLOR_MAP — Establishes new values for the color map.

FORMAT

GPR_$SET_COLOR_MAP (start-index, n-entries, values, status)

INPUT PARAMETERS

start-index
Index of first color value entry, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

n-entries
Number of entries. This is a 2-byte integer. Valid values are:

2, for monochromatic displays
1-16, for color displays in 4-bit pixel format
1-256, for color displays in 8-bit or 24-bit pixel format

values
Color value entries, in GPR_$COLOR_VECTOR_T format. This is an array of 4-byte integers.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* For the monochromatic display, the default start-index is 0, n-entries is 2, and the values are GPR_$BLACK and GPR_$WHITE. Dark has the value GPR_$BLACK, and bright has the value GPR_$WHITE. A program can use this routine to redefine the pixel values corresponding to bright and dark intensity.

* For the monochromatic display, if the program provides fewer than two values, or if the first two values are the same (both black or both white), the routine returns an error.

* For the monochromatic display, the graphics primitives simulate a color map by modifying the contents of display memory.

* In direct mode, you must acquire the display before establishing new values for the color map.

* To retrieve the current color map, use GPR_$INQ_COLOR_MAP.

GPR_$SET_COORDINATE_ORIGIN — Establishes x- and y-offsets to add to all x- and y-coordinates used as input to move, line drawing, and BLT operations on the current bitmap.

FORMAT

```
GPR_$SET_COORDINATE_ORIGIN (origin, status)
```

INPUT PARAMETERS

origin
: The new coordinate origin for the bitmap, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* This call affects the x- and y-coordinates input to the following routines: GPR_$MOVE, GPR_$LINE, GPR_$POLYLINE, GPR_$MULTILINE, GPR_$ADDITIVE_BLT, GPR_$BIT_BLT, and GPR_$PIXEL_BLT.
* To retrieve the current coordinate origin, use GPR_$INQ_COORDINATE_ORIGIN.
* The default coordinate origin is (0,0).

LIMITATIONS

* This routine may not be used on a bitmap corresponding to a Display Manager frame.

GPR_$SET_CURSOR_ACTIVE — Specifies whether the cursor is displayed.

FORMAT

```
GPR_$SET_CURSOR_ACTIVE (active, status)
```

INPUT PARAMETERS

active
: A Boolean (logical) value which specifies whether to display the cursor. Set the parameter to true to display the cursor; set it to false if you do not want to display the cursor.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Initially, the cursor is not displayed.
* To inquire whether the cursor is currently displayed, use GPR_$INQ_CURSOR.

LIMITATIONS

* A program may call this routine only while operating in borrow-display or direct mode.

GPR_$SET_CURSOR_ORIGIN — Defines one of the cursor's pixels as the cursor origin.

FORMAT

GPR_$SET_CURSOR_ORIGIN (origin, status)

INPUT PARAMETERS

origin
The position of one cursor pixel (the origin) relative to the entire cursor, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the pixel's cursor-relative x-coordinate. The second element is the pixel's cursor-relative y-coordinate. Coordinate values are in the range 0-15, but limited by cursor size; the cursor origin cannot be outside the cursor.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* A program uses GPR_$SET_CURSOR_ORIGIN to designate one cursor pixel as the cursor origin. Thereafter, when the cursor is moved, the pixel designated as the cursor origin moves to the screen coordinate designated as the cursor position.

* The default cursor origin depends on the default cursor size, which depends on the size of the Display Manager's standard font.

* To inquire about the current cursor origin, pattern, position and whether the cursor is enabled, use GPR_$INQ_CURSOR.

GPR_$SET_CURSOR_PATTERN — Loads a cursor pattern.

FORMAT

GPR_$SET_CURSOR_PATTERN (cursor-pattern, status)

INPUT PARAMETERS

cursor-pattern
: The descriptor of the bitmap which contains the cursor pattern, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Initially, the cursor pattern is a rectangle, which varies in size according to the size of the Display Manager's standard font. A program can use GPR_$SET_CURSOR_PATTERN to redefine the cursor pattern. The bitmap that represents the cursor pattern consists of one plane, which is a maximum of 16x16 pixels in size.

* To inquire about the current cursor pattern, use GPR_$INQ_CURSOR.

GPR_$SET_CURSOR_POSITION -- Establishes a position on the screen for display of the cursor.

FORMAT

```
GPR_$SET_CURSOR_POSITION (position, status)
```

INPUT PARAMETERS

position
Screen coordinate position for display of the cursor, in GPR_$POSITION_T format. This is a two-element array of two-byte integers. The first element is the cursor position's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the display in use, as follows:

	X	Y
Borrowed Display:		
Monochromatic Portrait:	0-799	0-1023
Monochromatic Landscape:	0-1023	0-799
Color:	0-1023	0-1023
Display Manager Frame:	0-32767	0-32767

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* Cursor position: If a program calls this routine when in borrow-display mode, the x- and y-coordinates represent an absolute position on the screen. If a program calls this routine when the cursor is inside a frame of a Display Manager pad, the x- and y-coordinates are offsets from the top left corner of the frame.

* If the coordinate position would cause any part of the cursor to be outside the screen or frame, the cursor moves only as far as the edge of the screen. The cursor is neither clipped nor made to disappear.

* To request the current cursor position, use GPR_$INQ_CURSOR.

LIMITATIONS

* In a Display Manager frame, this routine moves the cursor only if the cursor is in the window viewing this frame when the call is issued. If not, a "next window" command which moves to that window will move the cursor to its new position.

GPR_$SET_DRAW_VALUE -- Specifies the color/intensity value to use to draw lines.

FORMAT

```
GPR_$SET_DRAW_VALUE (index, status)
```

INPUT PARAMETERS

index
The color map index that indicates the current color/intensity value used for drawing lines, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0-1	for monochromatic displays
0-15	for color displays in 4-bit pixel format
0-255	for color displays in 8-bit or 24-bit pixel format
-2	for all displays. This specifies using the color/intensity value of the bitmap background as the line drawing value. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To retrieve the current draw value, use GPR_$INQ_DRAW_VALUE.
* The default draw value is 1.
* For monochromatic displays, only the low-order bit of the draw value is considered, because monochromatic displays have only one plane.
* For color displays in 4-bit pixel format, only the four lowest-order bits of the draw value are considered, because these displays have four planes.
* The color specification parameter is a color map index, not a color value. See Section 4.3.

GPR_$SET_FILL_BACKGROUND_VALUE — Specifies the color/intensity value used for drawing the background of tile fills.

FORMAT

```
GPR_$SET_FILL_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

index
: The color map index that indicates the current color/intensity value used for tile fills, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

  0-1 for monochromatic displays
  0-15 for color displays in 4-bit pixel format
  0-255 for color displays in 8-bit or 24-bit pixel format
  -1 for all displays. This specifies that the fill background is transparent; that is, the old values of the pixels are not changed.
  -2 for all displays. This specifies using the color/intensity value of the bitmap background as the fill background. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* To retrieve the current background value, use GPR_$INQ_FILL_BACKGROUND_VALUE.
* The default fill background value is -2.
* This routine defines the background fill value for 1-bit patterns. In all other fill patterns, the values set with this routine are ignored.

GPR_$SET_FILL_PATTERN — Specifies the fill pattern used for the current bitmap.

FORMAT

```
GPR_$SET_FILL_PATTERN (pattern, scale, status)
```

INPUT PARAMETERS

pattern
: The descriptor of the bitmap containing the fill pattern, in GPR_$BITMAP_DESC_T format. This is a 4-byte integer. See restriction below.

scale
: The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. See restriction below.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Currently, the tile pattern must be stored in a bitmap that is 32x32 pixels by n planes. The scale factor must be one. Any other pattern size or scale value results in an error.

* To retrieve the current fill pattern for the current bitmap, use GPR_$INQ_FILL_PATTERN.

* With a one-plane bitmap as the pattern, the pixel values used are those set by GPR_$SET_FILL_VALUE and GPR_$SET_FILL_BACKGROUND_VALUE. Pixels corresponding to "1" bits of the pattern are drawn in the fill value: pixels corresponding to "0" bits of the pattern are drawn in the fill background value.

* With a multiplane bitmap as the pattern, the pixel values used are those contained in the pattern bitmap.

* To re-establish solid fills, set the fill pattern descriptor to GPR_$NIL_BITMAP_DESC.

GPR_$SET_FILL_VALUE — Specifies the color/intensity value to use to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
GPR_$SET_FILL_VALUE (index, status)
```

INPUT PARAMETERS

index
: The color map index that indicates the current fill color/intensity value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

  0-1 for monochromatic displays
  0-15 for color displays in 4-bit pixel format
  0-255 for color displays in 8-bit or 24-bit pixel format

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.
* The default fill value is 1.
* For monochromatic displays, only the low-order bit of the fill value is considered, because monochromatic displays have only one plane.
* For color displays in 4-bit pixel format, only the four lowest-order bits of the fill value are considered, because these displays have four planes.
* The color specification parameter is a color map index, not a color value. See Section 4.3.

GPR_$SET_HORIZONTAL_SPACING — Specifies the parameter for horizontal spacing of the specified font.

FORMAT

```
GPR_$SET_HORIZONTAL_SPACING (font-id, horizontal-spacing, status)
```

INPUT PARAMETERS

font-id
: The identifier of the text font. This is a 2-byte integer.

horizontal-spacing
: The horizontal spacing parameter of the specified font. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Use GPR_$INQ_HORIZONTAL_SPACING to retrieve a font's horizontal spacing.
* The initial horizontal spacing is defined in the font file. For a description of fonts and font files, see the DOMAIN System Command Reference Manual.
* To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

GPR_$SET_IMAGING_FORMAT — Sets the imaging format of the color display.

FORMAT

GPR_$SET_IMAGING_FORMAT (format, status)

INPUT PARAMETERS

format
A format type that specifies pixel and bit values. Use GPR_$IMAGING_FORMAT_T format. This is a two-bute integer. Valid values and hardware configurations are the following:

Either two- or three-board:
GPR_$INTERACTIVE

Two-board only:
GPR_$IMAGING_1024x1024x8

Three-board only:
GPR_$IMAGING_512x512x24

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To retrieve the current imaging format, use GPR_$INQ_IMAGING_FORMAT.
* To use GPR_$SET_IMAGING_FORMAT, you must be in borrow display mode and be using a color node.
* See Section 2.3 for more information about imaging formats.

LIMITATIONS

* Imaging formats support only limited GPR operations — displaying pixel data and changing the color map. Other functions will return error messages.
* 1024x1024x8 imaging format is not supported on a three-board system as it offers no advantages over interactive formats.

GPR_$SET_INPUT_SID — Specifies the input pad from which graphics input is to be taken.

FORMAT

```
GPR_$SET_INPUT_SID (stream-id, status)
```

INPUT PARAMETERS

stream-id
: The stream ID that GPR software will use for input in frame mode, in STREAM_$ID_T format. The stream must be a Display Manager input pad.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* Programs use this call only when they call input routines in frame mode (GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT).
* If this routine is not called, the default stream ID is STREAM_$STDIN (a stream ID of zero).
* To work properly, the input pad must be the pad associated with the transcript pad passed to GPR_$INIT. STREAM_$STDIN is associated with STREAM_$STDOUT in this way in a normal Shell process window. Other process input pads derive their association from the PAD_$CREATE call that created them.

GPR_$SET_LINE_PATTERN — Specifies the pattern to use in drawing lines.

FORMAT

GPR_$SET_LINE_PATTERN (repeat, pattern, length, status)

INPUT PARAMETERS

repeat
: The replication factor for each bit in the pattern. This is a 2-byte integer.

pattern
: The bit pattern, left justified, in GPR_$LINE_PATTERN_T format. This is a four-element array of 2-byte integers.

length
: The length of the pattern in bits. This is a 2-byte integer in the range of 0 to 64.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$LINE, GPR_$POLYLINE, GPR_$MULTILINE use the pattern/style most recently defined by either GPR_$SET_LINE_PATTERN or GPR_$SET_LINESTYLE. The actual bits in the integers define the line pattern. You should set the first bit in the pattern; otherwise, the vectors you draw will not show the beginning of the line correctly.

* Specifying the value of 0 for either repeat or length results in a solid line.

* You may also set a line pattern with GPR_$SET_LINESTYLE. The pattern is defined by the parameter GPR_$DOTTED.

* Within each element of the bit pattern, the bits are used in order of decreasing significance. This starts with the most significant bit of entry 1 down to the least significant of entry 4.

* Use GPR_$INQ_LINE_PATTERN to retrieve the current line pattern. This routine returns the pattern set explicitly with GPR_$SET_LINE_PATTERN or set implicitly with GPR_$SET_LINESTYLE.

GPR_$SET_LINESTYLE — Sets the line-style attribute of the current bitmap.

FORMAT

GPR_$SET_LINESTYLE (style, scale, status)

INPUT PARAMETERS

style
The style of line, in GPR_$LINESTYLE_T format. Possible values are GPR_$SOLID and GPR_$DOTTED. This is a 2-byte integer.

scale
The scale factor for dashes if the style parameter is GPR_$DOTTED. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* When the line-style attribute is GPR_$DOTTED, lines are drawn in dashes. The scale factor determines the number of pixels in each dash and in each space between the dashes.
* For greater flexibility in setting line styles, use GPR_$SET_LINE_PATTERN.
* Use GPR_$INQ_LINESTYLE to retrieve the current line-style attribute.

GPR_$SET_OBSCURED_OPT — Establishes the action to be taken when a window to be acquired is obscured.

FORMAT

```
GPR_$SET_OBSCURED_OPT (if-obscured, status)
```

INPUT PARAMETERS

if-obscured
: If the window to be acquired by GPR_$ACQUIRE_DISPLAY is obscured, this argument specifies, in GPR_$OBSCURED_OPTION_T format, the action to be taken. This is a 2-byte integer. The possible actions are:

GPR_$POP_IF_OBS	Pop the window.
GPR_$ERR_IF_OBS	Return an error and do not acquire the display.
GPR_$BLOCK_IF_OBS	Block display acquisition until the window is popped.
GPR_$OK_IF_OBS	Acquire the display even though the window is obscured.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* If this routine is not called, the action to be taken defaults to GPR_$ERR_IF_OBS.
* These options apply whenever the display is acquired, either by GPR_$ACQUIRE_DISPLAY or implicitly by GPR_$EVENT_WAIT.
* If the program specifies the option GPR_$ERR_IF_OBS, it must check the status code returned from GPR_$ACQUIRE_DISPLAY or GPR_$EVENT_WAIT before calling any drawing routines.
* When a program specifies OK_$IF_OBS, the output is performed even when the window is obscured. This output may overwrite other Display Manager windows.
* Use GPR_$INQ_VIS_LIST to retrieve a list of visible sections of an obscured window.

GPR_$SET_PLANE_MASK — Establishes a plane mask for subsequent write operations.

FORMAT

```
GPR_$SET_PLANE_MASK (mask, status)
```

INPUT PARAMETERS

mask
The plane mask, which specifies which planes to use, in GPR_$MASK_T format. FORTRAN programmers should encode the plane mask in a 2-byte integer in the range 0-255. Pascal programmers can use a variable with a SET OF 0..7 to set the bits directly. If, for example, the third bit is ON, the third plane will be used in subsequent operations.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* The default mask specifies that all planes are used.

* Operations occur only on the planes specified in the mask. A program can use this routine, for example, to perform raster operations on separate planes or groups of planes in the bitmap.

* Using the mask, a program can partition the 8-bit pixels into subunits. For example, the program can use planes 0-3 for one picture and planes 4-7 for another. Thus, one bitmap may contain two color pictures. This does not, however, increase the number of colors available for one bitmap.

* To retrieve the current plane mask, use GPR_$INQ_CONSTRAINTS.

GPR_$SET_RASTER_OP — Specifies a new raster operation for both BLTs and lines.

FORMAT

GPR_$SET_RASTER_OP (plane-id, raster-op, status)

INPUT PARAMETERS

plane-id
Identifier of the bitmap plane involved in the raster operation, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are zero through the identifier of the bitmap's highest plane.

raster-op
Raster operation code, in GPR_$RASTER_OP_T format. This is a 2-byte integer. Possible values are zero through fifteen.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* See Table 3-1 for a listing of the raster operation codes and their functions.

* Use GPR_$INQ_RASTER_OPS to retrieve the current BLT raster operation.

* The initial raster operation is 3. This operation assigns all source bit values to new destination bit values.

* The following is a list of the op codes and logical functions of the sixteen raster operations and a truth table of the raster operations.

Raster Operations and Their Functions

Op Code	Logical Function
0	Assign zero to all new destination values.
1	Assign source AND destination to new destination.
2	Assign source AND complement of destination to new destination.
3	Assign all source values to new destination.
4	Assign complement of source AND destination to new destination.
5	Assign all destination values to new destination.
6	Assign source EXCLUSIVE OR destination to new destination.
7	Assign source OR destination to new destination.
8	Assign complement of source AND complement of destination to new destination.
9	Assign source EQUIVALENCE destination to new destination.
10	Assign complement of destination to new destination.
11	Assign source OR complement of destination to new destination.
12	Assign complement of source to new destination.
13	Assign complement of source OR destination to new destination.
14	Assign complement of source OR complement of destination to new destination.
15	Assign 1 to all new destination values.

Raster Operations: Truth Table

SOURCE BIT VALUE	DESTINATION BIT VALUE	RESULTANT BIT VALUES FOR THE FOLLOWING OP CODES: 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

GPR_$SET_REFRESH_ENTRY — Specifies the entry points of application-supplied procedures that refresh the displayed image in a direct window and hidden display memory.

FORMAT

```
GPR_$SET_REFRESH_ENTRY (window-procedure, disp-mem-procedure, status)
```

INPUT PARAMETERS

window-procedure
: Entry point for the application-supplied procedure that refreshes the Display Manager window, in GPR_$RWIN_PR_T format. This is a pointer to a procedure.

disp-mem-procedure
: Entry point for the application-supplied procedure that refreshes the application's hidden display memory, in GPR_$RHDM_PR_T format, which is a pointer to a procedure.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The Display Manager determines when the window needs to be redrawn based on the amount of activity the user generates on the screen. When a redrawing operation is necessary, the Display Manager calls the application-supplied procedure the next time that the application acquires the display. Two input parameters are passed to the window refresh procedure:

    - unobscured — When false, this Boolean value indicates that the window is obscured.

    - position_changed — When true, this Boolean value indicates that the window has moved or grown since the display was released.

  The "pointer to procedure" data type is an extension to the Pascal language. See the Pascal User's Guide for an explanation of this extension.

* This routine requires you to pass procedure pointers. To do so in FORTRAN programs, use the following technique. First declare the subroutines to be passed as EXTERNAL. Then pass their names using the IADDR function to simulate the Pascal pointer mechanism. For example:

```
EXTERNAL REFRESH_WINDOW, REFRESH_HDM

      .
      .
      .

CALL GPR_$SET_REFRESH_ENTRY (IADDR(REFRESH_WINDOW),IADDR(REFRESH_HDM),
                             STATUS)
```

* In FORTRAN, use 0 (not NIL) to indicate a zero value.

* The contents of hidden display memory may be altered by an intervening process. If this occurs, the application-supplied procedure for display memory refresh is called to reconstruct hidden display memory when the application reacquires the display (with GPR_$ACQUIRE_DISPLAY or GPR_$EVENT_WAIT).

* You may use GPR_$SET_REFRESH_ENTRY to specify entry points for window refresh and hidden display memory refresh procedures. When you do so, the display memory refresh procedure is called before the window refresh procedure. These procedures are never called asynchronously. They are called only when the display is acquired. Programs that only need to define one entry point should specify a null pointer to the appropriate parameter.

* Successive calls to GPR_$SET_REFRESH_ENTRY override previously defined entry points.

* This routine applies to the current bitmap. When you have multiple displayed bitmaps, you may establish a call to GPR_$SET_REFRESH_ENTRY for each bitmap. When a program changes attribute blocks for a bitmap during a graphics session, the refresh entry procedures are lost unless you set them for the new attribute block.

* Applications can also direct the Display Manager to refresh the window automatically; see the routine GPR_$SET_AUTO_REFRESH.

GPR_$SET_SPACE_SIZE — Specifies the size of horizontal spacing for the specified font.

FORMAT

```
GPR_$SET_SPACE_SIZE (font-id, space-size, status)
```

INPUT PARAMETERS

font-id
: Identifier of the text font. This is a 2-byte integer.

space-size
: The space size of the specified font. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* To retrieve a font's space size, use GPR_$INQ_SPACE_SIZE.

* The initial character widths are defined in the font file. For a description of fonts and font files, see the DOMAIN System Command Reference Manual.

* To use routines which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

* The space size is the number of pixels to skip in the horizontal direction when you include a character that is not in the font.

GPR_$SET_TEXT_BACKGROUND_VALUE -- Specifies the color/intensity value to use for text background.

FORMAT

```
GPR_$SET_TEXT_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

index
The color map index that indicates the current color/intensity value used for the text background, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0-1	for monochromatic displays
0-15	for color displays in 4-bit pixel format
0-255	for color displays in 8-bit or 24-bit pixel format
-1	for all displays. This specifies that the text background is transparent; that is, the old values of the pixels are not changed.
-2	for all displays. This specifies using the color/intensity value of the bitmap background as the text background. For borrowed displays and memory bitmaps, this value is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To retrieve the current text background value, use GPR_$INQ_VALUES,
* The default text background value is -2.
* For monochromatic displays, only the low-order bit of the text background value is considered, because monochromatic displays have only one plane.
* For color displays in 4-bit pixel mode, only the four lowest-order bits of the text background value are considered, because these displays have four planes.
* The color specification parameter is a color map index, not a color value. See Section 4.3.

GPR_$SET_TEXT_FONT -- Establishes a new font for subsequent text operations.

FORMAT

GPR_$SET_TEXT_FONT (font-id, status)

INPUT PARAMETERS

font-id
Identifier of the new text font. This is a 2-byte integer.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* Obtain the font-id when loading a font with GPR_$LOAD_FONT_FILE.
* To request the identifier of the current font, use GPR_$INQ_TEXT.
* There is no default text font. A program must load and set the font.
* Call GPR_$SET_TEXT_FONT for each main memory bitmap. Otherwise, an error is returned (invalid font id).

GPR_$SET_TEXT_PATH — Specifies the direction for writing a line of text.

FORMAT

GPR_$SET_TEXT_PATH (direction, status)

INPUT PARAMETERS

direction
    The direction used for writing text, in GPR_$DIRECTION_T format. Possible values are GPR_$UP, GPR_$DOWN, GPR_$LEFT and GPR_$RIGHT.

OUTPUT PARAMETERS

status
    Completion status, in STATUS_$T format.

NOTES

* To retrieve the current text path, use GPR_$INQ_TEXT_PATH.

* The initial text path is GPR_$RIGHT.

GPR_$SET_TEXT_VALUE -- Specifies the color/intensity value to use for writing text.

FORMAT

```
GPR_$SET_TEXT_VALUE (index, status)
```

INPUT PARAMETERS

index
The color map index that indicates the current color/intensity value used for writing text, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0-1 for monochromatic displays
0-15 for color displays in 4-bit pixel format
0-255 for color displays in 8-bit or 24-bit pixel format

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To retrieve the current text value, use GPR_$INQ_VALUES.
* The default text value is 1 for borrowed displays, memory bitmaps, and Display Manager frames on monochromatic displays; 0 for Display Manager frames on color displays.
* For monochromatic displays, only the low-order bit of the text value is considered, because monochromatic displays have only one plane.
* For color displays in 4-bit pixel format, only the four lowest-order bits of the text value are considered, because these displays have four planes.
* The color specification parameter is a color map index, not a color value. See Section 4.3.

GPR_$SET_WINDOW_ID — Establishes the character that identifies the current bitmap's window.

FORMAT

```
GPR_$SET_WINDOW_ID (character, status)
```

INPUT PARAMETERS

character
The character that identifies the current bitmaps's window. This is data type CHAR.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* This character is returned by GPR_$EVENT_WAIT and GPR_$COND_EVENT_WAIT when they return GPR_$ENTERED_WINDOW events. The character indicates which window was entered.

* The character 'A' is the default value of the window identification for all windows.

* You may assign the same character to more than one window. However, if you do so, you cannot distinguish input from the two windows.

GPR_$SPLINE_CUBIC_P — Draws a parametric cubic spline through the control points.

FORMAT

```
GPR_$SPLINE_CUBIC_P (x, y, npositions, status)
```

INPUT PARAMETERS

x
: List of the x-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits unless clipping is enabled.

npositions
: Number of coordinate positions. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$SPLINE_CUBIC_P draws a smooth curve starting from the current position, through each of the specified points.
* After the spline is drawn, the last point becomes the current position.
* The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.
* An error is returned if any two consecutive points are equal.
* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$SPLINE_CUBIC_X -- Draws a cubic spline as a function of x through the control points.

FORMAT

GPR_$SPLINE_CUBIC_X (x, y, npositions, status)

INPUT PARAMETERS

x
List of the x-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits unless clipping is enabled.

y
List of the y-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits unless clipping is enabled.

npositions
Number of coordinate positions. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$SPLINE_CUBIC_X draws a smooth curve starting from the current position and through each of the specified points.

* After the spline is drawn, the last point becomes the current position.

* The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

* An error is returned if any x-coordinate is less than or equal to a previous x-coordinate. The x-coordinate array must be sorted into increasing order.

* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$SPLINE_CUBIC_Y -- Draws a cubic spline as a function of y through the control points.

FORMAT

```
GPR_$SPLINE_CUBIC_Y (x, y, npositions, status)
```

INPUT PARAMETERS

x
: List of the x-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. This is a one-dimensional array of 2-byte integers. GPR_$COORDINATE_ARRAY_T format is an example of such an array. The values must be within the bitmap limits unless clipping is enabled.

npositions
: Number of coordinate positions. This is a 2-byte integer in the range 1-32767.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$SPLINE_CUBIC_Y draws a smooth curve starting from the current position and through each of the specified points.
* After the spline is drawn, the last point becomes the current position.
* The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.
* An error is returned if any y-coordinate is less than or equal to a previous y-coordinate. The y-coordinate array must be sorted into increasing order.
* When you have clipping enabled, you can specify coordinates outside the bitmap limits. With clipping disabled, specifying coordinates outside the bitmap limits results in an error.

GPR_$START_PGON — Defines the starting position of a polygon boundary loop of edges.

FORMAT

```
GPR_$START_PGON (x, y, status)
```

INPUT PARAMETERS

x
: The x-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

y
: The y-coordinate, in GPR_$COORDINATE_T format. This is a 2-byte integer. Its values must be within bitmap limits, unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* GPR_$START_PGON defines the first point in a polygon boundary loop of edges. This routine is used in conjunction with GPR_$PGON_POLYLINE to define a connected series of edges composing one closed loop of a polygon's boundary. To see the polygon, you must fill it using either GPR_$CLOSE_FILL_PGON or GPR_$CLOSE_RETURN_PGON and GPR_$MULTITRAPEZOID.

* This routine closes any previously open loop of edges by connecting its last endpoint to its first endpoint with an edge. Then, the routine starts the new loop.

* See Section 6.2 for more information about polygons.

GPR_$TERMINATE — Terminates the graphics primitives package.

FORMAT

GPR_$TERMINATE (delete-display, status)

INPUT PARAMETERS

delete-display
A Boolean (logical) value which specifies whether to delete the frame of the Display Manager pad. If the program has operated in a Display Manager frame and needs to delete the frame at the end of a graphics session, set this value to true. If the program needs to close, but not delete the frame, set this value to false. If the program has not used a Display Manager frame, the value is ignored.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$TERMINATE deletes the frame regardless of the value of the delete-display argument in the following case. A BLT operation from a memory bitmap has been done to a Display Manager frame since the last time GPR_$CLEAR was called for the frame.

GPR_$TEXT — Writes text to the current bitmap, beginning at the current position.

FORMAT

GPR_$TEXT (string, string-length, status)

INPUT PARAMETERS

string
The string to write, in GPR_$STRING_T format. This is an array of characters.

string-length
Number of characters in the string. This is a 2-byte integer. The maximum value is 256.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$TEXT always clips to the edge of the bitmap, regardless of whether clipping is enabled.

* GPR_$TEXT writes the characters in the current font which correspond to the ASCII values of the characters in the specified string. If the font does not have a character which corresponds to a character in the string, GPR_$TEXT leaves a space.

* Text is written at the current position. The origin of the first character of the character string is placed at the current position. Generally, the origin of the character is at the bottom left, exluding descenders of the character.

* Upon completion of the GPR_$TEXT routine, the current position is updated to the coordinate position where a next character would be written. This is the case even if the string is partly or completely clipped. However, the current position always remains within the boundaries of the bitmap.

GPR_$TRAPEZOID — Draws and fills a trapezoid.

FORMAT

GPR_$TRAPEZOID (trapezoid, status)

INPUT PARAMETERS

trapezoid
The trapezoid in the current bitmap to be filled in. Trapezoid is in GPR_$TRAP_T format, which is a two-element array in GPR_$HORIZ_SEG_T format. The first array element is the top horizontal segment of the trapezoid; the second array element is the bottom horizontal segment.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$TRAPEZOID fills in a trapezoid with the color/intensity value specified with GPR_$SET_FILL_VALUE. To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

* The GPR routines define a trapezoid as a quadrilateral with two horizontally parallel sides.

* See Section 6.2 for more information about filling trapezoids.

GPR_$TRIANGLE -- Fills a triangle.

FORMAT

GPR_$TRIANGLE (vertex-1, vertex-2, vertex-3, status)

INPUT PARAMETERS

vertex-1
The first vertex of the triangle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers describing an x- y-coordinate position in the bitmap.

vertex-2
The second vertex of the triangle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers describing an x- y-coordinate position in the bitmap.

vertex-3
The third vertex of the triangle, in GPR_$POSITION_T format. This is a two-element array of 2-byte integers describing an x- y-coordinate position in the bitmap.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* GPR_$TRIANGLE fills in a triangle with the color/intensity value specified with GPR_$SET_FILL_VALUE.

* To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

* See Section 6.2 for more information about filling triangles.

GPR_$UNLOAD_FONT_FILE — Unloads a font that has been loaded by GPR_$LOAD_FONT_FILE.

FORMAT

```
GPR_$UNLOAD_FONT_FILE (font-id, status)
```

INPUT PARAMETERS

font-id
: Font identifier. This is a 2-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The font-id is returned when a program loads a file with the routine GPR_$LOAD_FONT_FILE.

GPR_$WAIT_FRAME — Waits for the current frame refresh cycle to end before executing operations that modify the color display.

FORMAT

```
GPR_$WAIT_FRAME (status)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS

status
    Completion status, in STATUS_$T format.

NOTES

* This routine is for use on color displays only.
* Operations that modify the color display include block transfers, drawing lines, and writing text.
* This routine is useful primarily for animation. It delays execution of display modifications until the scan beam has completely covered the screen.
* A program can also use this routine to synchronize changes to the color map with the beginning of the frame.

GPR_$WRITE_PIXELS -- Writes the pixel values from a pixel array into a window of the current bitmap.

FORMAT

GPR_$WRITE_PIXELS (pixel-array, destination-window, status)

INPUT PARAMETERS

pixel-array
: Array from which to write pixel values (color/intensity). This is an array of 4-byte integers. GPR_$PIXEL_ARRAY_T format is an example of such an array.

destination-window
: Rectangular section of the current bitmap into which to write the pixel values, in GPR_$WINDOW_T format. This is a two-element array of two-element arrays. (In FORTRAN, you can use a two-dimensional array.)

  The first element of GPR_$WINDOW_T specifies the window start origin (top left corner), in GPR_$POSITION_T format. This is a two-element array of 2-byte integers. The first element is the origin's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the bitmap limits.

  The second element of GPR_$WINDOW_T specifies the window width and height, in GPR_$OFFSET_T format. This is a two-element array of 2-byte integers. The first element is the window width, in raster units; the second element is the window height, in raster units. Width and height values, when added to the window origin, must be within the bitmap limits.

OUTPUT PARAMETERS

status
: Completion status, in STATUS_$T format.

NOTES

* The pixel values in the pixel array, one in each 4-byte integer, are stored in the destination window of the bitmap in row-major order.

* For monochromatic displays, only the low-order bit of each pixel value is significant.

* For color displays in 4-bit pixel format, only the four lowest-order bits of each pixel value are considered because the bitmaps have four planes.

* GPR_$WRITE_PIXELS overwrites the old contents of the bitmap.

* To read pixel values from the current bitmap into an array, use GPR_$READ_PIXELS.

LIMITATIONS

* A program cannot use this routine on a bitmap corresponding to a Display Manager frame.

CHAPTER 12

GRAPHICS MAP FILES

A graphics map file, or GMF, is an image of the graphic information in a bitmap. Each bit in the GMF represents the state of one visible point on the display. On DOMAIN color nodes, where more than one plane is used to represent visible information, a GMF stores the state of only one plane, typically plane 0.

Once you have stored image data in a GMF, you can restore it to the display or produce a printed copy of the image. The GMF contains information that helps the GMF manager or application program interpret the contents of the GMF. For instance, the GMF may indicate the following: the physical density of the original image, whether "1" bits in the file correspond to light or dark points on the display, and the dimensions of the display area stored in the GMF. A GMF can contain the contents of an entire plane or any specified rectangular portion of the plane (subplane).

In Software Releases 6.0 and earlier, GMFs were called graphics metafiles.

The calls to the GMF manager begin with the letters GMF. To store image data in a GMF, you typically use GMF_$OPEN to create or open the GMF, then use GMF_$COPY_PLANE to specify the information to be copied into the GMF, then close the GMF with GMF_$CLOSE.

The GMF_$COPY_PLANE copies a plane of display memory. To make a GMF of any rectangular area on the display, regardless of its position, use the more general call GMF_$COPY_SUBPLANE.

The GMF_$RESTORE_PLANE call returns to the screen any image data that is stored in a specified GMF. To use this call, the node must be in borrow-display mode. The call changes a rectangular portion of the display, with the size determined by the size of the GMF you specify.

In place of graphic map files, it is recommended that you use external bitmap files. The external bitmap files are preferable in most cases. For a description of the routine for creating such files, see GPR_$OPEN_BITMAP_FILE in Chapter 11 of this manual.

12.1 INSERT FILES

Pascal programs using any GMF routines must include the file /SYS/INS/BASE.INS.PAS and /SYS/INS/GMF.INS.PAS. FORTRAN programs must include /SYS/INS/BASE.INS.FTN and /SYS/INS/GMF.INS.FTN. Programs written in C must include /SYS/INS/BASE.INS.C and /SYS/INS/GMF.INS.C.

12.2 DATA STRUCTURES

The GMF manager does not define any new data structures.

12.3 ERROR MESSAGES

Here are the possible error messages generated by the GMF calls described in this chapter:

```
GMF_$BAD_BPI     -- Bits/inch parameter is negative
GMF_$BAD_X_DIM   -- X dimension parameter is not positive
GMF_$BAD_Y_DIM   -- Y dimension parameter is not positive
GMF_$BAD_WPL     -- Words/line parameter is too small for the
                      X dimension you specified
GMF_$BAD_POS     -- Opening position parameter is illegal
GMF_$NOT_GMF     -- The file you wanted to open is not a GMF
```

12.4 PROGRAMMING EXAMPLE

This example in Pascal shows how to combine the calls described in this chapter with GPR calls (see Chapter 11) to form a typical GMF operation. This example restores a previously saved GMF.

```
    {Initialize the graphics primitive package in borrow-display mode.}

  gpr_$init (gpr_$borrow, 1, scsize, 0, disp_desc, sts);

    { Get the starting pointer. }

  gpr_$inq_bitmap_pointer (disp_desc, ptr, width, sts);

    { Open the file. }

  grf_$open ('//pepsi/adm/grf/turbine.pad',27,grf_$read,id,sts);

    { Restore the screen. }

  grf_$restore_plane (id,scsize.x_size,scsize.y_size,wpl,ptr,bpi,sts);
```

```
    { Close the file. }

  gmf_$close (id,status);
```

Explanation

The call to GPR_$INIT puts the screen in borrow-display mode. The next call obtains "ptr", the pointer to the start of the screen bitmap. The call to GMF_$OPEN opens a GMF with the specified name, returning the identification by which you subsequently refer to the GMF. The next call restores the screen from this GMF. (Alternatively, you can use a call here to copy a plane or subplane to the GMF.) The final call closes the GMF.

12.5 USER-CALLABLE ROUTINES

The remainder of this chapter specifies each call to the GMF manager in alphabetical order.

GMF_$CLOSE -- Closes a GMF

FORMAT

GMF_$CLOSE (stream-id, status)

INPUT PARAMETERS

stream-id
The stream-id of the GMF to be closed, in STREAM_$ID_T format. This is a 2-byte integer. You obtain the stream-id from the call to GMF_$OPEN that you used to open the GMF.

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To open a GMF, use GMF_$OPEN.

GMF_$COPY_PLANE -- Dumps a rectangular area of bits from virtual memory into a GMF.

FORMAT

```
GMF_$COPY_PLANE (stream-id, black-or-white, bpi, bit-pointer, x-dim, y-dim,
                 wpl, status)
```

INPUT PARAMETERS

stream-id
: The stream-id of the GMF into which the image is to be stored, in STREAM_$ID_T format. This is a 2-byte integer. You obtain the stream-id from the call to GMF_$OPEN that you used to open the GMF.

black-or-white
: A Boolean variable. A value of TRUE means "1" bits are black and "0" bits are white. A value of FALSE means "0" bits are white and "1" bits are black. This information is stored in the GMF.

bpi
: The number of bits per inch in the GMF. This information is also stored in the GMF. It indicates the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the image's original size. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image. This is a 2-byte integer.

bit-pointer
: A pointer to the upper left corner of the rectangular area to be stored, in GMF_$MEMORY_PTR_T format. This is a 4-byte integer. You obtain this value by calling the routine GPR_$INQ_BITMAP_POINTER.

x-dim
: The x dimension of the rectangular area to be stored in the GMF.

y-dim
: The y dimension of the rectangular area to be stored in the GMF.

wpl
: The number of 16-bit words per scan line in the GMF. The value of this parameter, which is stored in the GMF, is usually 64. By specifying a smaller value, you can produce a GMF that takes less storage space. However, the wpl must be at least 1/16 of the specified x-dim. For instance, if you are storing an area 400 bits wide in a GMF, the GMF must use at least 25 words to represent each scan line (row of dots).

OUTPUT PARAMETERS

status
    Completion status, in STATUS_$T format.

NOTES

* To store an image in a GMF, you must have opened the GMF with the GMF_$OPEN call.

* After storing an image in a GMF, close the GMF with the GMF_$CLOSE call.

* The GMF_$COPY_PLANE call is a special case of the GMF_$COPY_SUBPLANE call.

GMF_$COPY_SUBPLANE -- Dumps a rectangular area of bits from virtual memory into a GMF.

FORMAT

GMF_$COPY_SUBPLANE (stream-id, black-or-white, bpi, bit-pointer, x-dim, y-dim, x-offset, y-offset, wpl, status)

INPUT PARAMETERS

stream-id
The stream-id of the GMF into which the image is to be stored, in STREAM_$ID_T format. This is a 2-byte integer. You obtain the stream-id from the call to GMF_$OPEN that you used to open the GMF.

black-or-white
A Boolean variable. A value of TRUE means "1" bits are black and "0" bits are white. A value of FALSE means "0" bits are white and "1" bits are black. This information is stored in the GMF.

bpi
The number of bits per inch in the GMF. This information is also stored in the GMF. It indicates the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the image's original size. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image.

bit-pointer
A pointer to the upper left corner of the rectangular area to be stored. This is a four-byte integer. You obtain this value by calling the routine GPR_$INQ_BITMAP_POINTER.

x-dim
The x dimension of the rectangular area to be stored in the GMF.

y-dim
The y dimension of the rectangular area to be stored in the GMF.

x-offset
The x starting position of the rectangular area to be stored in the GMF.

y-offset
The y starting position of the rectangular area to be stored in the GMF.

wpl
The number of 16-bit words per scan line in the GMF. The value of this parameter, which is stored in the GMF, is usually 64. By specifying a smaller value, you can produce a GMF that takes less storage space. However, the wpl must be at least 1/16 of the specified x-dim. For instance, if you are storing an area 400 bits wide in a GMF, the GMF must use at least 25 words to represent each scan line (row of dots).

OUTPUT PARAMETERS

status
Completion status, in STATUS_$T format.

NOTES

* To copy a plane into a GMF, you must have opened the GMF with the GMF_$OPEN call.

* After copying a plane into a GMF, close the GMF with the GMF_$CLOSE call.

* The GMF_$COPY_SUBPLANE call is a more general form of the GMF_$COPY_PLANE call.

GMF_$OPEN -- Opens or creates a GMF.

FORMAT

GMF_$OPEN (name, name-length, start, stream-id, status)

INPUT PARAMETERS

name
Pathname, in NAME_$PNAME_T format.

name-length
The length of the name. This is a 2-byte integer.

start
Desired position in the file after open, in GMF_$OPOS_T format. This is a 2-byte integer. If you are opening the GMF to write data to it (to copy a plane or subplane into it), use one of these two constants:

GMF_$APPEND -- sets the initial position to EOF.
GMF_$OVERWRITE -- truncates the object to length 0 and sets the initial position to the beginning.

If you are opening the GMF to read data from it (restoring a plane), use this constant:

GMF_$READ -- sets the initial position to the beginning without truncating the GMF.

If the specified GMF does not exist and you used GMF_$OPEN to create it, it does not matter what value this parameter has.

OUTPUT PARAMETERS

stream-id
The stream-id of the opened GMF, in STREAM_$ID_T format. This is a 2-byte integer. You use this value in subsequent GMF calls that refer to the opened GMF.

status
Completion status, in STATUS_$T format.

NOTES

* If the specified GMF does not exist, the call to GMF_$OPEN creates it.

* You must call GMF_$OPEN before trying to read or write a GMF.

* After opening a GMF with GMF_$OPEN, you must eventually close it by calling GMF_$CLOSE.

GMF_$RESTORE_PLANE -- Copies an image back to the screen from a GMF.

FORMAT

GMF_$RESTORE_PLANE (stream-id, x-dim, y-dim, wpl, start, bpi, status)

INPUT PARAMETERS

stream-id
The stream-id of the GMF which is to supply the image, in STREAM_$ID_T format. This is a 2-byte integer. You obtain this parameter from the call to GMF_$OPEN you used to open the GMF.

x-dim
The x-dimension in bits of the display to which an image is to be restored. This is a 2-byte integer.

y-dim
The y-dimension in bits of the display to which an image is to be restored. This is a 2-byte integer.

wpl
The number of 16-bit words in the x-dimension in the destination bitmap. This is a 2-byte integer.

start
The starting address in the destination bitmap.

OUTPUT PARAMETERS

bpi
Bits per inch as specified in GMF_$COPY_PLANE.

status
Completion status, in STATUS_$T format.

NOTES

* Before calling GMF_$RESTORE_PLANE, you must use GPR_$INIT to place the node in borrow-display mode.

* The size of the area to be restored is the same as the size of the area you originally copied into the GMF. This information is contained in the GMF.

* The area to be restored is determined by the bit-pointer specified in the GMF_$RESTORE_PLANE call and the size data in the GMF. If this area runs off the right side or the bottom of the screen, the GMF manager restores only the portion of the stored image that fits on the screen.

* To restore a plane from a GMF, you must have opened the GMF with the GMF_$OPEN call.

* After restoring a plane from a GMF, you should close the GMF with the GMF_$CLOSE call.

APPENDIX A

KEYBOARD CHARTS

The following two charts and figures give the 8-bit ASCII values generated for two DOMAIN keyboards: 880 and low-profile. These charts include characters used in keystroke events. The columns represent the four highest order bits of an 8-bit value. The rows represent the four lowest order bits of an 8-bit value. For a more complete description of conventions for naming keys, see the DOMAIN System Command Reference Manual.

Figure A-1. Low-Profile keyboard Chart - Translated (user mode)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	^SP	^P	SP	0	@	P	`	p		R1		R1U	F1	F1S	F1U	F1C
1	^A	^Q	!	1	A	Q	a	q	L1	R2	L1U	R2U	F2	F2S	F2U	F2C
2	^B	^R	"	2	B	R	b	r	L2	R3	L2U	R3U	F3	F3S	F3U	F3C
3	^C	^S	#	3	C	S	c	s	L3	R4	L3U	R4U	F4	F4S	F4U	F4C
4	^D	^T	$	4	D	T	d	t	L4	R5	L4U	R5U	F5	F5S	F5U	F5C
5	^E	^U	%	5	E	U	e	u	L5	BS	L5U	R2S	F6	F6S	F6U	F6C
6	^F	^V	&	6	F	V	f	v	L6	CR	L6U	R3S	F7	F7S	F7U	F7C
7	^G	^W	'	7	G	W	g	w	L7	TAB	L7U	R4S	F8	F8S	F8U	F8C
8	^H	^X	(	8	H	X	h	x	L8	STAB	L8U	R5S	R1S	L8S	L1A	L1AU
9	^I	^Y	)	9	I	Y	i	y	L9	CTAB	L9U		L1S	L9S	L2A	L2AU
A	^J	^Z	*	:	J	Z	j	z	LA		LAU		L2S	LAS	L3A	L3AU
B	^K	ESC	+	;	K	[	^k	{	LB		LBU		L3S	LBS	R6	R6U
C	^L	^\	,	<	L	\	l	|	LC		LCU		L4S	LCS	L1AS	
D	^M	^]	-	=	M	]	m	}	LD		LDU		L5S	LDS	L2AS	
E	^N	^~	.	>	N	^	n		LE		LEU		L6S	LES	L3AS	
F	^O	^?	/	?	O		o	DEL	LF		LFU		L7S	LFS	R6S	

Figure A-2. Low-Profile Keyboard

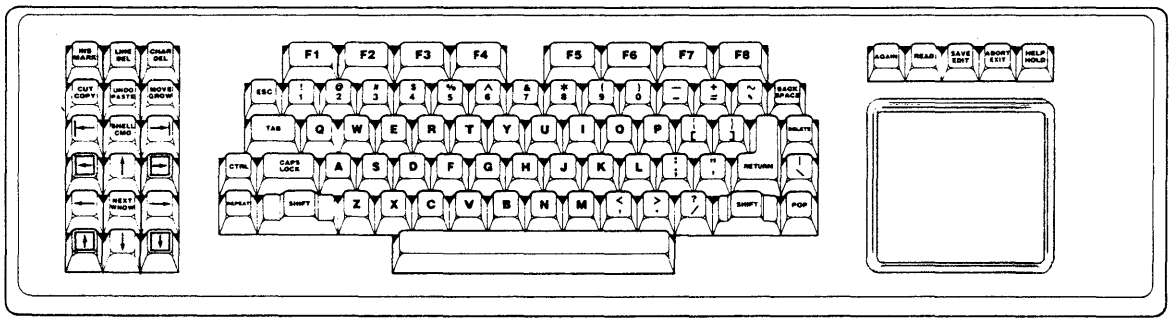


Figure A-3. 880 Keyboard

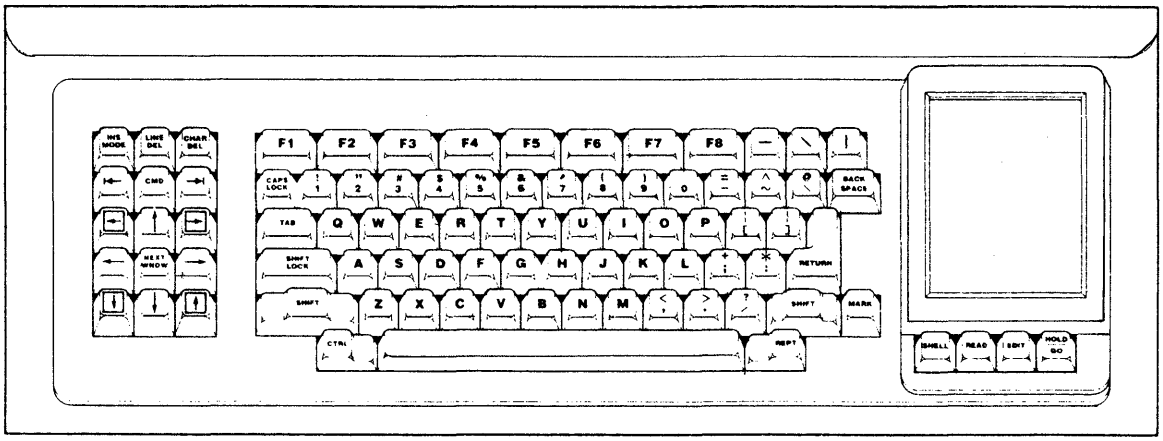


Figure A-4. 880 Keyboard Chart - Translated (user mode)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	^`	^P	SP	0	@	P	`	p		R1		R1U	F1	F1S	F1U	F1C
1	^A	^Q	!	1	A	Q	a	q	L1	R2	L1U	R2U	F2	F2S	F2U	F2C
2	^B	^R	"	2	B	R	b	r	L2	R3	L2U	R3U	F3	F3S	F3U	F3C
3	^C	^S	#	3	C	S	c	s	L3	R4	L3U	R4U	F4	F4S	F4U	F4C
4	^D	^T	$	4	D	T	d	t	L4	R5	L4U	R5U	F5	F5S	F5U	F5C
5	^E	^U	%	5	E	U	e	u	L5	BS	L5U		F6	F6S	F6U	F6C
6	^F	^V	&	6	F	V	f	v	L6	CR	L6U		F7	F7S	F7U	F7C
7	^G	^W	'	7	G	W	g	w	L7	TAB	L7U		F8	F8S	F8U	F8C
8	^H	^X	(	8	H	X	h	x	L8	STAB	L8U		N0	N8	N0U	N8U
9	^I	^Y	)	9	I	Y	i	y	L9	CTAB	L9U		N1	N9	N1U	N9U
A	^J	^Z	*	:	J	Z	j	z	LA		LAU		N2	N.	N2U	N.U
B	^K	^[	+	;	K	[	k	{	LB		LBU		N3	N=	N3U	N=U
C	^L	^\	,	<	L	\	l	|	LC		LCU		N4	N+	N4U	N+U
D	^M	^]	-	=	M	]	m	}	LD		LDU		N5	N-	N5U	N-U
E	^N	^~	.	>	N	^	n	~	LE		LEU		N6	N*	N6U	N*U
F	^O	^/	/	?	O		o	^	LF		LFU		N7	N/	N7U	N/U

APPENDIX B

INSERT FILE FOR PASCAL

This appendix contains the insert file for Pascal. This file includes error messages, color values, data types, and data structures. For a listing of data types and data structures in a table format, see Chapter 9.

```
CONST
        gpr_$operation_ok              = 16 00000000;
        gpr_$not_initialized           = 16 06010001;
        gpr_$already_initialized       = 16 06010002;
        gpr_$wrong_display_hardware    = 16 06010003;
        gpr_$illegal_for_frame         = 16 06010004;
        gpr_$must_borrow_display       = 16 06010005;
        gpr_$no_attributes_defined     = 16 06010006;
        gpr_$no_more_space             = 16 06010007;
        gpr_$dimension_too_big         = 16 06010008;
        gpr_$dimension_too_small       = 16 06010009;
        gpr_$bad_bitmap                = 16 0601000A;
        gpr_$bad_attribute_block       = 16 0601000B;
        gpr_$window_out_of_bounds      = 16 0601000C;
        gpr_$source_out_of_bounds      = 16 0601000D;
        gpr_$dest_out_of_bounds        = 16 0601000E;
        gpr_$invalid_plane             = 16 0601000F;
        gpr_$cant_deallocate           = 16 06010010;
        gpr_$coord_out_of_bounds       = 16 06010011;
        gpr_$invalid_color_map         = 16 06010012;
        gpr_$invalid_raster_op         = 16 06010013;
        gpr_$bitmap_is_read_only       = 16 06010014;
        gpr_$internal_error            = 16 06010015;
        gpr_$font_table_full           = 16 06010016;
        gpr_$bad_font_file             = 16 06010017;
        gpr_$invalid_font_id           = 16 06010018;
        gpr_$window_obscured           = 16 06010019;
        gpr_$not_in_direct_mode        = 16 0601001A;
        gpr_$not_in_polygon            = 16 0601001B;
        gpr_$kbd_not_acq               = 16 0601001C;
        gpr_$display_not_acq           = 16 0601001D;
        gpr_$illegal_pixel_values      = 16 0601001E;
```

```
        gpr_$illegal_when_imaging   = 16 0601001F;
        gpr_$invalid_imaging_format = 16 06010020;
        gpr_$must_release_display   = 16 06010021;
        gpr_$cant_mix_modes         = 16 06010022;
        gpr_$no_input_enabled       = 16 06010023;
        gpr_$duplicate_points       = 16 06010024;
        gpr_$array_not_sorted       = 16 06010025;
        gpr_$character_not_in_font  = 16 06010026;
        gpr_$illegal_fill_pattern   = 16 06010027;
        gpr_$illegal_fill_scale     = 16 06010028;
        gpr_$incorrect_alignment    = 16 06010029;
        gpr_$illegal_text_path      = 16 0601002A;
        gpr_$unable_to_rotate_font  = 16 0601002B;
        gpr_$font_is_read_only      = 16 0601002C;
        gpr_$illegal_pattern_length = 16 0601002D;

CONST
        gpr_$black       = 0;          { color value for black }
        gpr_$white       = 16 FFFFFF;  { color value for white }
        gpr_$red         = 16 FF0000;  { color value for red }
        gpr_$green       = 16 00FF00;  { color value for green }
        gpr_$blue        = 16 0000FF;  { color value for blue }
        gpr_$cyan        = 16 00FFFF;  { color value for cyan (blue + green) }
        gpr_$magenta     = 16 FF00FF;  { color value for magenta (red + blue) }
        gpr_$yellow      = 16 FFFF00;  { color value for yellow (red + green) }

        gpr_$transparent = -1;         { pixel value for transparent
                                         (no change) }
        gpr_$background  = -2;         { pixel value for window background }

        gpr_$string_size = 256;        { number of chars in a gpr string }
        gpr_$max_x_size = 4096;        { max   bits in bitmap x dimension }
        gpr_$max_y_size = 4096;        { max   bits in bitmap y dimension }
        gpr_$highest_plane = 7;        { max plane number in a bitmap }
        gpr_$nil_attribute_desc = 0;   { value of a descriptor of
                                         nonexistent attributes }
        gpr_$nil_bitmap_desc = 0;      { value of a descriptor of a
                                         nonexistent bitmap }
        gpr_$max_bmf_group = 0;        { max group   in external bitmaps }
        gpr_$bmf_major_version = 1;
        gpr_$bmf_minor_version = 1;

TYPE
            { the five ways to use this package }
        gpr_$display_mode_t = (gpr_$borrow,
                               gpr_$frame,
                               gpr_$no_display,
                               gpr_$direct,
                               gpr_$borrow_nc
                              );

            { possible display hardware configurations }
        gpr_$display_config_t = (gpr_$bw_800x1024,
```

```
                            gpr_$bw_1024x800,
                            gpr_$color_1024x1024x4,
                            gpr_$color_1024x1024x8,
                            gpr_$RESERVEDx4,
                            gpr_$RESERVEDx8
                           );

    { imaging vs interactive display formats }
gpr_$imaging_format_t = (gpr_$interactive,
                            gpr_$imaging_1024x1024x8,
                            gpr_$imaging_512x512x24
                           );

    { input event types }
gpr_$event_t = ( gpr_$keystroke, gpr_$buttons, gpr_$locator,
                 gpr_$entered_window, gpr_$left_window,
                 gpr_$locator_stop, gpr_$no_event );

    { sets of key values for input events }
gpr_$keyset_t = SET OF char;

    { options for acquire-display behavior when window is obscured }
gpr_$obscured_opt_t = ( gpr_$ok_if_obs, gpr_$error_if_obs,
                        gpr_$pop_if_obs, gpr_$block_if_obs );

    { eventcount keys }
gpr_$ec_key_t = (gpr_$input_ec);

    { procedure type for refresh-window procedures }
gpr_$rwin_pr_t = ^PROCEDURE( IN unobscured: boolean;
                             IN pos_change: boolean );

    { procedure type for refresh-hidden display memory procedures }
gpr_$rhdm_pr_t = ^PROCEDURE;

    { bitmap coordinates }
gpr_$coordinate_t = integer16;

    { lists of bitmap coordinates }
gpr_$coordinate_array_t = ARRAY [1..10] OF gpr_$coordinate_t;

    { bitmap positions }
gpr_$position_t = RECORD
    x_coord, y_coord: gpr_$coordinate_t
    END;

    { bitmap offsets }
gpr_$offset_t = RECORD
    x_size, y_size: gpr_$coordinate_t;
    END;

    { windows on a bitmap }
gpr_$window_t = RECORD
```

```
    window_base: gpr_$position_t;
    window_size: gpr_$offset_t
    END;

    { lists of windows }
gpr_$window_list_t = ARRAY[1..10] OF gpr_$window_t;

    { horizontal line segments }
gpr_$horiz_seg_t = RECORD
    x_coord_l, x_coord_r, y_coord: gpr_$coordinate_t
    END;

    { trapezoids with horizontal bases:  defined as 2 horizontal
      line segments, top and bottom }
gpr_$trap_t = RECORD
    top, bot: gpr_$horiz_seg_t
    END;

    { lists of trapezoids with horizontal bases }
gpr_$trap_list_t = ARRAY [1..10] OF gpr_$trap_t;

    { graphics primitive strings }
gpr_$string_t = ARRAY [1..gpr_$string_size] OF char ;

    { bitmap plane numbers }
gpr_$plane_t = 0..gpr_$highest_plane;

    { bitmap plane masks }
gpr_$mask_t = SET OF gpr_$plane_t;

    { color values }
gpr_$color_t = linteger;

    { pixel values }
gpr_$pixel_value_t = linteger;

    { arrays of color values }
gpr_$color_vector_t = ARRAY [0..255] OF gpr_$color_t;

    { arrays of pixel values }
gpr_$pixel_array_t = ARRAY [0..131072] OF gpr_$pixel_value_t;

    { raster operation opcodes }
gpr_$raster_op_t = 0..15;

    { arrays of raster operation opcodes }
gpr_$raster_op_array_t = ARRAY [gpr_$plane_t] OF gpr_$raster_op_t;

    { scalar type for directions }
gpr_$direction_t = (gpr_$up, gpr_$down, gpr_$left, gpr_$right) ;

    { line drawing styles }
gpr_$linestyle_t = (gpr_$solid, gpr_$dotted);
```

```
        gpr_$line_pattern_t = ARRAY [1..4] OF integer;

            { attribute block descriptors }
        gpr_$attribute_desc_t = linteger;

            { bitmap descriptors }
        gpr_$bitmap_desc_t = linteger;

            { external bitmap header version number }
        gpr_$version_t =
            RECORD
            major    : integer16;
            minor    : integer16
            END;

            { external bitmap group header descriptor }
        gpr_$bmf_group_header_t =
            RECORD
            n_sects          : integer16;
            pixel_size       : integer16;
            allocated_size   : integer16;
            bytes_per_line   : integer16;
            bytes_per_sect   : integer32;
            storage_offset   : univ_ptr;
            END;

            { array of external bitmap group header descriptors }
        gpr_$bmf_group_header_array_t =
            ARRAY [0..gpr_$max_bmf_group] OF gpr_$bmf_group_header_t;

            { ways to access external bitmap objects }
        gpr_$access_mode_t = (gpr_$create,gpr_$update, gpr_$write,
                              gpr_$readonly);
%eject;
{  Initialization and termination.  }


{  GPR_$INIT initializes the graphics primitive package.  }

PROCEDURE   gpr_$init (
                IN  op: gpr_$display_mode_t; { mode of operation }
                IN  unit_or_pad: stream_$id_t; { display unit or stream id
                                                 of dm pad if any }
                IN  size: gpr_$offset_t;    { disp bitmap sizes in bits for
                                              x and y }
                IN  hi_plane: gpr_$plane_t; { highest plane number display
                                              bitmap is to have }
                OUT init_bitmap: gpr_$bitmap_desc_t;    { descriptor of initial
                                                          bitmap }
                OUT status: status_$t       { returned status }
            ); EXTERN;
```

```
{  GPR_$TERMINATE terminates this package's operation.  }

PROCEDURE   gpr_$terminate (
                IN  delete_disp: boolean;    { whether to delete dm frame }
                OUT status: status_$t        { returned status }
            ); EXTERN;
%eject;
{  Set and inquire operations for the display.  }


{  GPR_$INQ_CONFIG returns the current display configuration.  }

PROCEDURE   gpr_$inq_config (
                OUT config: gpr_$display_config_t;
                OUT status: status_$t        { returned status }
            ); EXTERN;


{  GPR_$SET_COLOR_MAP gives new values for the color map.  }

PROCEDURE   gpr_$set_color_map (
                IN  start_index: gpr_$pixel_value_t;{ index of first entry }
                IN  n_entries: integer;      {   of entries }
                IN  v: UNIV gpr_$color_vector_t; { entry values }
                OUT status: status_$t        { returned status }
            ); EXTERN;


{  GPR_$INQ_COLOR_MAP returns current values in the color map.  }

PROCEDURE   gpr_$inq_color_map (
                IN  start_index: gpr_$pixel_value_t;{ index of first entry }
                IN  n_entries: integer;      {   of entries }
                OUT v: UNIV gpr_$color_vector_t;{ entry values }
                OUT status: status_$t        { returned status }
            ); EXTERN;


{  GPR_$SET_CURSOR_PATTERN loads a cursor pattern.  }

PROCEDURE   gpr_$set_cursor_pattern (
                IN  cursor: gpr_$bitmap_desc_t;{ pattern for cursor }
                OUT status: status_$t        { returned status }
            ); EXTERN;
```

```
{  GPR_$SET_CURSOR_ACTIVE specifies whether the cursor should be
   on or off.  }

PROCEDURE   gpr_$set_cursor_active (
                IN  active: boolean;        { false --> off, true --> on }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$SET_CURSOR_POSITION gives the position at which the cursor is
   to be displayed, in the current bitmap.  }

PROCEDURE   gpr_$set_cursor_position (
                IN  pos: gpr_$position_t;   { where it goes }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$SET_CURSOR_ORIGIN gives the cursor-relative position of its pixel
   which is to be placed at the cursor position.  }

PROCEDURE   gpr_$set_cursor_origin (
                IN  origin: gpr_$position_t; { the position of the cursor
                                               pixel }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$INQ_CURSOR returns information about the cursor.  }

PROCEDURE   gpr_$inq_cursor (
                OUT cursor: gpr_$bitmap_desc_t;{ pattern for cursor }
                OUT ops: gpr_$raster_op_array_t;{ raster ops for cursor }
                OUT active: boolean;        { whether cursor is active }
                OUT pos: gpr_$position_t;   { current cursor position }
                OUT origin: gpr_$position_t;{ current cursor origin }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$WAIT_FRAME causes the color display hardware to defer processing
   further requests until the next end-of-frame.  }

PROCEDURE   gpr_$wait_frame (
                OUT status: status_$t       { returned status }
            ); EXTERN;
%eject;
{  Bitmap control functions.  }
```

```
{  GPR_$SET_BITMAP establishes the current bitmap for subsequent
   operations.  }

PROCEDURE   gpr_$set_bitmap (
                IN  bitmap: gpr_$bitmap_desc_t;{ the given bitmap }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$INQ_BITMAP returns a descriptor of the current bitmap.  }

PROCEDURE   gpr_$inq_bitmap (
                OUT bitmap: gpr_$bitmap_desc_t;{ the current bitmap }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$ALLOCATE_BITMAP allocates a bitmap in main memory.  }

PROCEDURE   gpr_$allocate_bitmap (
                IN  size: gpr_$offset_t;    { sizes in bits for x and y }
                IN  hi_plane: gpr_$plane_t; { highest plane number bitmap is
                                              to have }
                IN  attr: gpr_$attribute_desc_t;{ attributes it is to have }
                OUT bitmap: gpr_$bitmap_desc_t;{ the bitmap returned }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$ALLOCATE_BITMAP_NC allocates a bitmap in main memory without
   zeroing it.  }

PROCEDURE   gpr_$allocate_bitmap_nc (
                IN  size: gpr_$offset_t;    { sizes in bits for x and y }
                IN  hi_plane: gpr_$plane_t; { highest plane number bitmap
                                              is to have }
                IN  attr: gpr_$attribute_desc_t;{ attributes it is to have }
                OUT bitmap: gpr_$bitmap_desc_t;{ the bitmap returned }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$ALLOCATE_HDM_BITMAP allocates a bitmap in hidden_display memory.  }

PROCEDURE   gpr_$allocate_hdm_bitmap (
                IN  size: gpr_$offset_t;    { sizes in bits for x and y }
                IN  hi_plane: gpr_$plane_t; { highest plane number bitmap is
                                              to have }
```

```
            IN  attr: gpr_$attribute_desc_t; { attributes it is to have }
            OUT bitmap: gpr_$bitmap_desc_t; { the bitmap returned }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$OPEN_BITMAP_FILE obtains access to an external bitmap }

PROCEDURE gpr_$open_bitmap_file (
            IN     access     : gpr_$access_mode_t;
            IN     filename   : UNIV name_$pname_t;   { pathname of bitmap
                                                        file }
            IN     namesize   : integer;              { length of file name }
            IN OUT version    : gpr_$version_t;
            IN OUT size       : gpr_$offset_t;         { x,y size of bitmap }
            IN OUT groups     : integer;
            IN OUT g_headers  : gpr_$bmf_group_header_array_t;
            IN     attribs    : gpr_$attribute_desc_t;{ attributes bitmap is
                                                        to have }
            OUT    bitmap     : gpr_$bitmap_desc_t;   { returned bitmap
                                                        descriptor   }
            OUT    created    : boolean;              { TRUE if GPR created
                                                        the file  }
            OUT    status     : status_$t
        );EXTERN;


{  GPR_$DEALLOCATE_BITMAP deallocates a bitmap allocated by
   ALLOCATE_BITMAP.  }

PROCEDURE   gpr_$deallocate_bitmap (
            IN  bitmap: gpr_$bitmap_desc_t;  { the bitmap to kill }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$INQ_BITMAP_POINTER returns a pointer to the storage/display memory
   for the given bitmap and the number of words each scan line occupies.
   This information can be used to directly manipulate the bits in the bitmap.  }

PROCEDURE   gpr_$inq_bitmap_pointer (
            IN  bitmap: gpr_$bitmap_desc_t; { the given bitmap }
            OUT storage_ptr: univ_ptr;  { the address of storage }
            OUT line_width: integer;    { number of 16-bit words per line }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$INQ_BM_BIT_OFFSET returns the number of bits in the most significant
   part of the first word of each scanline which are not part of the
   given bitmap.   In other words, the offset is the number of bits between
```

```
   a 16-bit word boundary and the left edge of the bitmap.
   Currently, this number can only be nonzero for direct graphics bitmaps.  }

PROCEDURE   gpr_$inq_bm_bit_offset (
                IN  bitmap: gpr_$bitmap_desc_t; { the given bitmap }
                OUT bit_offset: integer;    { in the range 0 to 15 }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$SELECT_COLOR_FRAME selects whether frame 0 (top 1024 lines) or
   frame 1 (bottom 1024 lines) is visible.  Normally frame 0 is visible.  }

PROCEDURE   gpr_$select_color_frame (
                IN  frame: integer;         { 0 or 1 }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$REMAP_COLOR_MEMORY sets the plane of frame 0 of color display memory
   (normally visible) which is mapped at the address returned by
    INQ_BITMAP_POINTER.  }

PROCEDURE   gpr_$remap_color_memory (
                IN  plane: gpr_$plane_t;    { plane for access thru storage_ptr }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$REMAP_COLOR_MEMORY_1 sets the plane of frame 1 of color display
   memory (normally hidden) which is mapped at the address returned by
   INQ_BITMAP_POINTER.  }

PROCEDURE   gpr_$remap_color_memory_1 (
                IN  plane: gpr_$plane_t;    { plane for access thru storage_ptr }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$COLOR_ZOOM sets the zoom scale factors for the color display.  }

PROCEDURE   gpr_$color_zoom (
                IN  x, y: integer;          { 1 to 16 }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$ENABLE_DIRECT_ACCESS waits for display hardware to finish current
```

```
   operations so that the user can access display memory directly.  }

PROCEDURE   gpr_$enable_direct_access (
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$SET_BITMAP_DIMENSIONS changes the size and number of planes
   of the given bitmap.  }

PROCEDURE   gpr_$set_bitmap_dimensions (
                IN  bitmap: gpr_$bitmap_desc_t; { the given bitmap }
                IN  size: gpr_$offset_t;    { new sizes in bits for x and y }
                IN  hi_plane: gpr_$plane_t; { new highest plane number for
                                              bitmap }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$INQ_BITMAP_DIMENSIONS returns the size and number of planes
   of the given bitmap.  }

PROCEDURE   gpr_$inq_bitmap_dimensions (
                IN  bitmap: gpr_$bitmap_desc_t; { the given bitmap }
                OUT size: gpr_$offset_t;    { sizes in bits for x and y }
                OUT hi_plane: gpr_$plane_t; { highest plane number bitmap has }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$ALLOCATE_ATTRIBUTE_BLOCK allocates an attribute block,
   initialized to default settings.  }

PROCEDURE   gpr_$allocate_attribute_block (
                OUT attrib: gpr_$attribute_desc_t; { new attribute block }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$DEALLOCATE_ATTRIBUTE_BLOCK deallocates an attribute block allocated
   by ALLOCATE_ATTRIBUTE_BLOCK.  }

PROCEDURE   gpr_$deallocate_attribute_block (
                IN  attrib: gpr_$attribute_desc_t; { attribute block to be
                                                     deleted }
                OUT status: status_$t       { returned status }
            ); EXTERN;
```

```
{  GPR_$SET_ATTRIBUTE_BLOCK establishes the given attributes as the attributes
   of the current bitmap.  }

PROCEDURE   gpr_$set_attribute_block (
                IN  attrib: gpr_$attribute_desc_t;{ new current attribute
                                                    block }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$ATTRIBUTE_BLOCK returns as the function value a descriptor of
   the attributes of the given bitmap.  }

FUNCTION    gpr_$attribute_block (
                IN  bitmap: gpr_$bitmap_desc_t;{ the bitmap with the
                                                 attributes wanted }
                OUT status: status_$t
            ): gpr_$attribute_desc_t; EXTERN;
%eject;
{  Set and inquire operations for bitmap attributes.  }


{  GPR_$SET_attribute sets an attribute in the current bitmap.
   The list of calls follows.  }

PROCEDURE   gpr_$set_clip_window (
                IN  window: gpr_$window_t;  { new clipping window }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_clipping_active (
                IN  active: boolean;        { false --> off, true --> on }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_text_font (
                IN  font_id: integer;       { new text font }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_text_path (
                IN path: gpr_$direction_t;  { text path }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_coordinate_origin (
                IN  origin: gpr_$position_t;{ new coordinate origin }
                OUT status: status_$t       { returned status }
            ); EXTERN;
```

```
PROCEDURE   gpr_$set_plane_mask (
                IN  mask: gpr_$mask_t;        { new plane mask }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_draw_value (
                IN  val: gpr_$pixel_value_t;{ new line-drawing value }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_text_value (
                IN  val: gpr_$pixel_value_t;{ new text value }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_text_background_value (
                IN  val: gpr_$pixel_value_t;{ new text background value }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_fill_value (
                IN  val: gpr_$pixel_value_t;{ new fill value }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_fill_background_value (
                IN  val: gpr_$pixel_value_t;{ new fill background value }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_fill_pattern (
                IN  pattern: gpr_$bitmap_desc_t; { bitmap containing tile }
                IN  scale: integer;           { scale factor for tile }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_raster_op (
                IN  plane: gpr_$plane_t;      { plane for new raster op }
                IN  op: gpr_$raster_op_t;     { new raster op }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_linestyle (
                IN  style: gpr_$linestyle_t;{ new line style }
                IN  scale: integer;           { scale factor for dashes }
                OUT status: status_$t         { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_line_pattern (
                IN  repeat_count: integer;  { scale factor }
                IN  pattern: gpr_$line_pattern_t;
                IN  length: integer;          { number of pattern bits }
                OUT status: status_$t         { returned status }
```

```
        ); EXTERN;

PROCEDURE   gpr_$set_character_width (
                IN  font_id: integer;       { font_id }
                IN  character: char;        { character to modify }
                IN  width: integer;         { new width }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_horizontal_spacing (
                IN  font_id: integer;       { font_id }
                IN  horizontal_spacing: integer; { new horiz-spacing }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$set_space_size (
                IN  font_id: integer;       { font_id }
                IN  space_size: integer;    { new space-size }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$INQ_attributes returns the current settings of a group
   of attributes for the current bitmap.  }

PROCEDURE   gpr_$inq_constraints (
                OUT window: gpr_$window_t;  { clipping window }
                OUT active: boolean;        { whether clipping is active }
                OUT mask: gpr_$mask_t;      { plane mask }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_text (
                OUT font_id: integer;       { font id }
                OUT path: gpr_$direction_t; { text path }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_text_path (
                OUT path: gpr_$direction_t; { text path }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_coordinate_origin (
                OUT origin: gpr_$position_t;{ the coordinate origin }
                OUT status: status_$t       { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_draw_value (
                OUT val: gpr_$pixel_value_t;{ line-drawing value }
                OUT status: status_$t       { returned status }
            ); EXTERN;
```

```
PROCEDURE   gpr_$inq_text_values (
                OUT tval: gpr_$pixel_value_t;{ text value }
                OUT bval: gpr_$pixel_value_t;{ text background value }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_fill_value (
                OUT val: gpr_$pixel_value_t;{ fill value }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_fill_background_value (
                OUT val: gpr_$pixel_value_t;{ new fill background value }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_fill_pattern (
                OUT pattern: gpr_$bitmap_desc_t; { bitmap containing tile }
                OUT scale: integer;          { scale factor for tile }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_raster_ops (
                OUT ops: gpr_$raster_op_array_t; { raster ops for all
                                                   planes }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_linestyle (
                OUT style: gpr_$linestyle_t;{ line style }
                OUT scale: integer;          { scale factor for dashes }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_line_pattern (
                OUT repeat_count: integer;  { scale factor }
                OUT pattern: gpr_$line_pattern_t;
                OUT length: integer;         { number of pattern bits }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_character_width (
                IN  font_id: integer;        { font id}
                IN  character: char;         { specified character }
                OUT width: integer;          { width of character }
                OUT status: status_$t        { returned status }
            ); EXTERN;

PROCEDURE   gpr_$inq_horizontal_spacing (
                IN  font_id: integer;        { font id}
                OUT horizontal_spacing: integer; { horiz spacing }
                OUT status: status_$t        { returned status }
            ); EXTERN;
```

```
PROCEDURE   gpr_$inq_space_size (
                IN  font_id: integer;       { font id}
                OUT space_size: integer;    { space size }
                OUT status: status_$t       { returned status }
            ); EXTERN;
%eject;
{  Drawing operations.  }


{  GPR_$MOVE sets the CP to the given position.  }

PROCEDURE   gpr_$move (
                IN  x, y: gpr_$coordinate_t;{ where to go }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$INQ_CP returns the current position.  }

PROCEDURE   gpr_$inq_cp (
                OUT x, y: gpr_$coordinate_t;{ x and y of current position }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$LINE draws a line from the CP to the given position
   and sets the CP to the given position.  }

PROCEDURE   gpr_$line (
                IN  x, y: gpr_$coordinate_t;{ where to draw line to }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$POLYLINE does a series of LINEs, starting from the CP.  }

PROCEDURE   gpr_$polyline (
                IN  x, y: UNIV gpr_$coordinate_array_t; { the list of
                                                          endpoints }
                IN  npoints: integer;       { how many }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$MULTILINE does a series of alternate MOVEs and LINEs. }

PROCEDURE   gpr_$multiline (
```

```
            IN  x, y: UNIV gpr_$coordinate_array_t; { the list of
                                                      endpoints }
            IN  npoints: integer;       { how many }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$DRAW_BOX draws a rectangular box  }

PROCEDURE   gpr_$draw_box (
            IN  x1,y1,x2,y2 : gpr_$coordinate_t; { corners }
            OUT status : status_$t { returned status }
        ); EXTERN;


{  GPR_$ARC_3P draw an arc from current point through two points p2 and p3. }

PROCEDURE   gpr_$arc_3p (
            IN  p2: gpr_$position_t;    { second point on the arc }
            IN  p3: gpr_$position_t;    { third point on the arc }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$CIRCLE draws a circle of radius around point center. }

PROCEDURE   gpr_$circle (
            IN  center: gpr_$position_t;{ center of circle }
            IN  radius: integer;        { radius of circle }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$CIRCLE_FILLED draws a filled circle of radius around point center. }

PROCEDURE   gpr_$circle_filled (
            IN  center: gpr_$position_t;{ center of circle }
            IN  radius: integer;        { radius of circle }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$SPLINE_CUBIC_P draws a parametric cubic spline through the
   control points. }

PROCEDURE   gpr_$spline_cubic_p (
            IN  x, y: UNIV gpr_$coordinate_array_t; { list of control
                                                      points }
            IN  npoints: integer;       { number of points }
```

```
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$SPLINE_CUBIC_X draws a cubic spline as a function of x through the
  control points. }

PROCEDURE   gpr_$spline_cubic_x (
            IN  x, y: UNIV gpr_$coordinate_array_t; { list of control
                                                      points }
            IN  npoints: integer;       { number of points }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$SPLINE_CUBIC_Y draws a cubic spline as a function of y through
  the control points. }

PROCEDURE   gpr_$spline_cubic_y (
            IN  x, y: UNIV gpr_$coordinate_array_t; { list of control
                                                      points }
            IN  npoints: integer;       { number of points }
            OUT status: status_$t       { returned status }
        ); EXTERN;
%eject;
{ Text operations. }


{ GPR_$LOAD_FONT_FILE loads a font contained in a file into an appro-
  priate area (based on the current display mode and configuration). }

PROCEDURE   gpr_$load_font_file (
            IN  pn: UNIV name_$pname_t; { pathname of file }
            IN  pnlen: integer;         { pathname length }
            OUT font_id: integer;       { returned font id }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$UNLOAD_FONT_FILE unloads a font that has been loaded by
  LOAD_FONT_FILE. }

PROCEDURE   gpr_$unload_font_file (
            IN  font_id: integer;       { font id, from load_font_file }
            OUT status: status_$t       { returned status }
        ); EXTERN;
```

```
{  GPR_$TEXT writes text to a bitmap from the current position ("CP").  }

PROCEDURE   gpr_$text (
                IN  str: UNIV gpr_$string_t;     { the string to write }
                IN  strl: integer;               { how long it is }
                OUT status: status_$t            { returned status }
            ); EXTERN;


{  GPR_$INQ_TEXT_EXTENT returns the x- and y-offsets the given string
   would span if written with TEXT.  }

PROCEDURE   gpr_$inq_text_extent (
                IN  str: UNIV gpr_$string_t;     { the string to be inquired
                                                   about }
                IN  strl: integer;               { how long it is }
                OUT size: gpr_$offset_t;         { how big it would be }
                OUT status: status_$t            { returned status }
            ); EXTERN;


{  GPR_$INQ_TEXT_OFFSET returns the x- and y-offsets that must be added
   to the coordinates of the desired upper left pixel of the string to
   give the pixel from which the TEXT call should be made, and the
   x- or y-offset of the pixel which would be the updated CP.  }

PROCEDURE   gpr_$inq_text_offset (
                IN  str: UNIV gpr_$string_t;     { the string to be inquired
                                                   about }
                IN  strl: integer;               { how long it is }
                OUT start: gpr_$offset_t;        { where it would start relative
                                                   to upper left pixel of string }
                OUT xy_end: integer;             { where next write would start
                                                   relative to same in x or y}
                OUT status: status_$t            { returned status }
            ); EXTERN;


{  GPR_$REPLICATE_FONT creates and loads a read/write copy of the original
   font.  }

PROCEDURE   gpr_$replicate_font (
                IN  font_id: integer;            { original font id }
                OUT repl_font_id: integer;       { replicated font id }
                OUT status: status_$t            { returned status }
            ); EXTERN;
%eject;
{  Data transfer operations.  }
```

```
{  GPR_$CLEAR clears the current bitmap to a given pixel value.  }

PROCEDURE   gpr_$clear (
                IN  val: gpr_$pixel_value_t;{ value to set whole bitmap to }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$READ_PIXELS reads the pixels from the given window of the current
   bitmap and stores them in a pixel array.  }

PROCEDURE   gpr_$read_pixels (
                IN  src_w: gpr_$window_t;   { the source window }
                OUT pix: UNIV gpr_$pixel_array_t;{ the pixel array }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$WRITE_PIXELS writes the pixels from a pixel array into the given
   window of the current bitmap.  }

PROCEDURE   gpr_$write_pixels (
                IN  pix: UNIV gpr_$pixel_array_t;{ the pixel array }
                IN  dst_w: gpr_$window_t;   { the destination window }
                OUT status: status_$t       { returned status }
            ); EXTERN;
%eject;
{  BLT operations.  }


{  GPR_$PIXEL_BLT moves a rectangle of whole pixels from the source
   bitmap to a position in the current bitmap.  }

PROCEDURE   gpr_$pixel_blt (
                IN  src_b: gpr_$bitmap_desc_t;{ the source bitmap }
                IN  src_w: gpr_$window_t;   { the source window }
                IN  dst_o: gpr_$position_t; { the destination origin
                                              (in current bitmap) }
                OUT status: status_$t       { returned status }
            ); EXTERN;


{  GPR_$BIT_BLT moves a rectangle of bits from a plane of the source
   bitmap to a position in a plane of the current bitmap.  }

PROCEDURE   gpr_$bit_blt (
                IN  src_b: gpr_$bitmap_desc_t;{ the source bitmap }
                IN  src_w: gpr_$window_t;   { the source window }
                IN  src_p: gpr_$plane_t;    { the source plane }
```

```
            IN  dst_o: gpr_$position_t; { the destination origin }
            IN  dst_p: gpr_$plane_t;    { the destination plane }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$ADDITIVE_BLT moves a rectangle of bits from a plane of the source
  bitmap to a position in every plane of the current bitmap. }

PROCEDURE   gpr_$additive_blt (
            IN  src_b: gpr_$bitmap_desc_t;{ the source bitmap }
            IN  src_w: gpr_$window_t;   { the source window }
            IN  src_p: gpr_$plane_t;    { the source plane }
            IN  dst_o: gpr_$position_t; { the destination origin }
            OUT status: status_$t       { returned status }
        ); EXTERN;
%eject;
{ Fill operations. }


{ GPR_$RECTANGLE fills a rectangle in the current bitmap. }

PROCEDURE   gpr_$rectangle (
            IN  rect: gpr_$window_t;    { the rectangle to fill }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$TRAPEZOID fills a trapezoid in the current bitmap. }

PROCEDURE   gpr_$trapezoid (
            IN  trap: gpr_$trap_t;      { the trapezoid to fill }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$MULTITRAPEZOID fills a list of trapezoids in the current bitmap. }

PROCEDURE   gpr_$multitrapezoid (
            IN  t_list: UNIV gpr_$trap_list_t;  { the trapezoids to fill }
            IN  n_traps: integer;       { how many }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{ GPR_$TRIANGLE fills a triangle in the current bitmap. }

PROCEDURE   gpr_$triangle (
            IN  p1, p2, p3: gpr_$position_t;  { vertices of the triangle }
```

```
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$START_PGON starts a polygon boundary loop.  }

PROCEDURE   gpr_$start_pgon (
            IN  x, y: gpr_$coordinate_t;{ the first point in the loop }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$PGON_POLYLINE defines a series of line segments as part of
   the current polygon boundary loop.  }

PROCEDURE   gpr_$pgon_polyline (
            IN  x, y: UNIV gpr_$coordinate_array_t; { the list of
                                                      endpoints }
            IN  npoints: integer;       { how many }
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$CLOSE_FILL_PGON closes and fills the currently open polygon.  }

PROCEDURE   gpr_$close_fill_pgon (
            OUT status: status_$t       { returned status }
        ); EXTERN;


{  GPR_$CLOSE_RETURN_PGON closes the currently open polygon,
   decomposes it, and returns the list of trapezoids.  }

PROCEDURE   gpr_$close_return_pgon (
            IN  list_size: integer;     { how many trapezoids t_list
                                          can hold }
            OUT t_list: UNIV gpr_$trap_list_t;  { the trapezoid list }
            OUT n_traps: integer;       { how many trapezoids in the
                                          whole decomposition }
            OUT status: status_$t       { returned status }
        ); EXTERN;
%eject;
{  Direct graphics calls.  }


{  GPR_$SET_ACQ_TIME_OUT sets the acquire timeout for a display in
   direct mode.  }
```

```
PROCEDURE gpr_$set_acq_time_out(
                IN time_out : time_$clock_t; { expected maxtime that window
                                               will be in use }
                OUT sts : status_$t          { returned status }
            ); EXTERN;


{  GPR_SET_OBSCURED_OPT sets obscured window behavior for acquiring a
   display in direct mode.  }

PROCEDURE gpr_$set_obscured_opt(
                IN if_obscured: gpr_$obscured_opt_t; { one of four options }
                OUT sts: status_$t                   { returned status }
            ); EXTERN;


{  GPR_$ACQUIRE_DISPLAY gives the user exclusive access to all display
   operations in the acquired window, returning whether window was unobscured.  }

FUNCTION gpr_$acquire_display(
                OUT sts: status_$t  { returned status }
            ): boolean; EXTERN;


{  GPR_$INQ_VIS_LIST returns list of visible subwindows when a window is
   obscured.  }

PROCEDURE gpr_$inq_vis_list(
                IN slots_avail: integer;     { number of subwin to return }
                OUT slots_total: integer;    { number of subwin that exist }
                OUT vis_list: UNIV gpr_$window_list_t; { list of vis subwindows }
                OUT sts: status_$t           { returned status }
             ); EXTERN;


{  GPR_$FORCE_RELEASE releases the display regardless of how many times it
   was previously acquired.  }

PROCEDURE gpr_$force_release(
                OUT acq_rel_cnt: integer;    { number of times window was
                                               acquired }
                OUT sts: status_$t           { returned status }
            ); EXTERN;


{  GPR_$RELEASE_DISPLAY decrements the display-acquired count.  }

PROCEDURE gpr_$release_display(
```

```
                OUT sts: status_$t      { returned status }
            ); EXTERN;


{  GPR_$SET_REFRESH_ENTRY provides two procedures which will refresh the
   window and refresh hidden display memory.  }

PROCEDURE gpr_$set_refresh_entry(
                IN rwin_proc: gpr_$rwin_pr_t;   { entry point for refreshing
                                                  window }
                IN rhdm_proc: gpr_$rhdm_pr_t;   { entry point for hidden
                                                  display memory }
                OUT sts: status_$t          { returned status }
            ); EXTERN;


{  GPR_$SET_AUTO_REFRESH tells the display mangager whether to save
   window's contents and refresh screen when needed.  }

PROCEDURE gpr_$set_auto_refresh(
                IN auto_refresh: boolean;  { on or off }
                OUT sts: status_$t         { returned status }
            ); EXTERN;
%eject;
{  Input calls.  }


{  GPR_$EVENT_WAIT waits for input or timeout; returns boolean to indicate
   unobscured window.  }

FUNCTION gpr_$event_wait(
                OUT event_type: gpr_$event_t; { type of event that occurred }
                OUT event_data: char;         { char associated with the event }
                OUT pos: gpr_$position_t;     { x and y position }
                OUT sts: status_$t            { returned status }
            ): boolean; EXTERN;


{  GPR_$COND_EVENT_WAIT is like GPR_$EVENT_WAIT but returns immediately.  }

FUNCTION gpr_$cond_event_wait(
                OUT event_type: gpr_$event_t; { type of event that occurred }
                OUT event_data: char;         { char associated with the event }
                OUT pos: gpr_$position_t;     { x and y position }
                OUT sts: status_$t            { returned status }
            ): boolean; EXTERN;
```

```
{  GPR_$ENABLE_INPUT enables the given event type and set of keys to be
   recognized by event wait calls.  }

PROCEDURE gpr_$enable_input(
                IN event_type: gpr_$event_t; { specified event type }
                IN key_set: gpr_$keyset_t;   { set of char }
                OUT sts: status_$t           { returned status }
            ); EXTERN;


{  GPR_$DISABLE_INPUT disables the given event type.  }

PROCEDURE gpr_$disable_input(
                IN event_type: gpr_$event_t; { specified event type to
                                               disable }
                OUT sts: status_$t           { returned status }
            ); EXTERN;


{  GPR_$GET_EC gets the gpr input eventcount.  }

PROCEDURE gpr_$get_ec(
                IN gpr_key: gpr_$ec_key_t;   { specifies which eventcount
                                               to obtain }
                OUT ec_ptr: ec2_$ptr_t;      { event count address }
                OUT sts: status_$t           { returned status }
            ); EXTERN;


{  GPR_$SET_INPUT_SID establishes the stream id for gpr input in frame mode
   if not standard input.  }

PROCEDURE   gpr_$set_input_sid (
                IN sid: stream_$id_t;        { stream id of input source }
                OUT sts: status_$t           { returned status }
             ); EXTERN;


{  GPR_$SET_WINDOW_ID sets the character identifying the current displayed
   bitmap for input identification purposes.  }

PROCEDURE   gpr_$set_window_id (
                IN  id_char: char;           { character to identify window }
                OUT sts: status_$t           { returned status }
             ); EXTERN;


{  GPR_$INQ_WINDOW_ID returns the character identifying the current displayed
```

```
   bitmap for input identification purposes.  }

PROCEDURE   gpr_$inq_window_id (
                OUT id_char: char;          { character to identify window }
                OUT sts: status_$t          { returned status }
             ); EXTERN;
%eject;
{  Imaging format calls.  }


{  GPR_$SET_IMAGING_FORMAT sets the DN600 display format.  }

PROCEDURE   gpr_$set_imaging_format (
                IN  format: gpr_$imaging_format_t;  { the imaging format }
                OUT status: status_$t               { returned status }
            ); EXTERN;


{  GPR_$INQ_IMAGING_FORMAT returns the current DN600 display format.  }

PROCEDURE   gpr_$inq_imaging_format (
                OUT format: gpr_$imaging_format_t;  { the current imaging
                                                      format }
                OUT status: status_$t               { returned status }
            ); EXTERN;

%eject;
```

APPENDIX C

INSERT FILE FOR C

This appendix contains the insert file for C. This file includes error messages, color values, data types, and data structures. For a listing of data types and data structures in a table format, see Chapter 9.

```
define gpr_$operation_ok 0x00000000
define gpr_$not_initialized 0x06010001
define gpr_$already_initialized 0x06010002
define gpr_$wrong_display_hardware 0x06010003
define gpr_$illegal_for_frame 0x06010004
define gpr_$must_borrow_display 0x06010005
define gpr_$no_attributes_defined 0x06010006
define gpr_$no_more_space 0x06010007
define gpr_$dimension_too_big 0x06010008
define gpr_$dimension_too_small 0x06010009
define gpr_$bad_bitmap 0x0601000A
define gpr_$bad_attribute_block 0x0601000B
define gpr_$window_out_of_bounds 0x0601000C
define gpr_$source_out_of_bounds 0x0601000D
define gpr_$dest_out_of_bounds 0x0601000E
define gpr_$invalid_plane 0x0601000F
define gpr_$cant_deallocate 0x06010010
define gpr_$coord_out_of_bounds 0x06010011
define gpr_$invalid_color_map 0x06010012
define gpr_$invalid_raster_op 0x06010013
define gpr_$bitmap_is_read_only 0x06010014
define gpr_$internal_error 0x06010015
define gpr_$font_table_full 0x06010016
define gpr_$bad_font_file 0x06010017
define gpr_$invalid_font_id 0x06010018
define gpr_$window_obscured 0x06010019
define gpr_$not_in_direct_mode 0x0601001A
define gpr_$not_in_polygon 0x0601001B
define gpr_$kbd_not_acq 0x0601001C
define gpr_$display_not_acq 0x0601001D
```

```
define gpr_$illegal_pixel_values 0x0601001E
define gpr_$illegal_when_imaging 0x 0601001F
define gpr_$invalid_imaging_format 0x06010020
define gpr_$must_release_display 0x06010021
define gpr_$cant_mix_modes 0x06010022
define gpr_$no_input_enabled 0x06010023
define gpr_$duplicate_points 0x06010024
define gpr_$array_not_sorted 0x06010025
define gpr_$character_not_in_font 0x06010026
define gpr_$illegal_fill_pattern 0x06010027
define gpr_$illegal_fill_scale 0x06010028
define gpr_$incorrect_alignment 0x06010029
define gpr_$illegal_text_path 0x0601002A
define gpr_$unable_to_rotate_font 0x0601002B
define gpr_$font_is_read_only 0x0601002C
define gpr_$illegal_pattern_length 16 0601002D

define gpr_$black 0                     /* color value for black */
define gpr_$white 0xFFFFFF              /* color value for white */
define gpr_$red 0xFF0000                /* color value for red */
define gpr_$green 0x00FF00              /* color value for green */
define gpr_$blue 0x0000FF               /* color value for blue */
define gpr_$cyan 0x00FFFF               /* color value for cyan
                                           (blue + green) */
define gpr_$magenta 0xFF00FF            /* color value for magenta (red + blue) */
define gpr_$yellow 0xFFFF00             /* color value for yellow (red + green) */

define gpr_$transparent (-1)            /* pixel value for transparent
                                           (no change) */
define gpr_$background  (-2)            /* pixel value for window background */

define gpr_$string_size 256             /* number of chars in a gpr string */
define gpr_$max_x_size 4096             /* max   bits in bitmap x dimension */
define gpr_$max_y_size 4096             /* max   bits in bitmap y dimension */
define gpr_$highest_plane 7             /* max plane number in a bitmap */
define gpr_$nil_attribute_desc 0        /* value of a descriptor of
                                           nonexistent attributes */
define gpr_$nil_bitmap_desc 0           /* value of a descriptor of a
                                           nonexistent bitmap */

define gpr_$max_bmf_group 0             /* max group   in external bitmaps */
define gpr_$bmf_major_version 1
define gpr_$bmf_minor_version 1

typedef enum { gpr_$keystroke, gpr_$buttons, gpr_$locator,
               gpr_$entered_window, gpr_$left_window,
               gpr_$locator_stop, gpr_$no_event
} gpr_$event_t;

unsigned long gpr_$keyset_t[4];    /* This is supposed to be SET OF char */


typedef enum { gpr_$ok_if_obs, gpr_$error_if_obs,
```

```
                gpr_$pop_if_obs, gpr_$block_if_obs
} gpr_$obscured_opt_t;

typedef enum {gpr_$input_ec} gpr_$ec_key_t;

typedef void (*gpr_$rwin_pr_t)();
typedef void (*gpr_$rhdm_pr_t)();

            /* the five ways to use this package */
typedef enum {
        gpr_$borrow,
        gpr_$frame,
        gpr_$no_display,
        gpr_$direct,
        gpr_$borrow_nc
} gpr_$display_mode_t;

            /* possible display hardware configurations */
typedef enum {
        gpr_$bw_800x1024,
        gpr_$bw_1024x800,
        gpr_$color_1024x1024x4,
        gpr_$color_1024x1024x8,
        gpr_$RESERVEDx4,
        gpr_$RESERVEDx8
} gpr_$display_config_t;

            /* imaging vs interactive display formats */
typedef enum {
         gpr_$interactive,
         gpr_$imaging_1024x1024x8,
         gpr_$imaging_512x512x24
} gpr_$imaging_format_t;

            /* bitmap coordinates */
typedef short gpr_$coordinate_t;

            /* lists of bitmap coordinates */
typedef gpr_$coordinate_t gpr_$coordinate_array_t[10];

            /* bitmap positions */
typedef struct {
            gpr_$coordinate_t x_coord, y_coord;
} gpr_$position_t;

            /* bitmap offsets */
typedef struct {
            unsigned short x_size, y_size;
} gpr_$offset_t;

            /* windows on a bitmap */
typedef struct {
            gpr_$position_t window_base;
```

```
            gpr_$offset_t window_size;
} gpr_$window_t;

typedef gpr_$window_t gpr_$window_list_t[10];

            /* horizontal line segments */
typedef struct {
            gpr_$coordinate_t x_coord_l, x_coord_r, y_coord;
} gpr_$horiz_seg_t;

            /* trapezoids with horizontal bases:  defined as 2 horizontal
              line segments, top and bottom */
typedef struct {
            gpr_$horiz_seg_t top, bot;
} gpr_$trap_t;

            /* lists of trapezoids with horizontal bases */
typedef gpr_$trap_t gpr_$trap_list_t[10];

            /* graphics primitive strings */
typedef char gpr_$string_t[gpr_$string_size];

            /* bitmap plane numbers */
typedef unsigned short gpr_$plane_t;

            /* bitmap plane masks (bit vector) */
typedef short gpr_$mask_t;

            /* color values */
typedef linteger gpr_$color_t;

            /* pixel values */
typedef linteger gpr_$pixel_value_t;

            /* arrays of color values */
typedef gpr_$color_t gpr_$color_vector_t[256];

            /* arrays of pixel values */
typedef gpr_$pixel_value_t gpr_$pixel_array_t[131073];

            /* raster operation opcodes */
typedef unsigned short gpr_$raster_op_t;

            /* arrays of raster operation opcodes */
typedef gpr_$raster_op_t gpr_$raster_op_array_t[8];

            /* scalar type for directions */
typedef enum {gpr_$up, gpr_$down, gpr_$left, gpr_$right} gpr_$direction_t;

            /* line drawing styles */
typedef enum {gpr_$solid, gpr_$dotted} gpr_$linestyle_t;

typedef short gpr_$line_pattern_t[4];
```

```
            /* attribute block descriptors */
typedef linteger gpr_$attribute_desc_t;

            /* bitmap descriptors */
typedef linteger gpr_$bitmap_desc_t;

typedef struct {
    unsigned short major,minor;
} gpr_$version_t;


typedef struct {
    unsigned short n_sects;
    unsigned short pixel_size;
    unsigned short allocated_size;
    unsigned short bytes_per_line;
    linteger       bytes_per_sect;
    integer       *storage_offset;
} gpr_$bmf_group_header_t;

typedef gpr_$bmf_group_header_t gpr_$bmf_group_header_array_t
[gpr_$max_bmf_group+1];

typedef enum {gpr_$create,gpr_$update, gpr_$write, gpr_$readonly}
gpr_$access_mode_t;
 eject
/*  Initialization and termination.  */


/*  GPR_$INIT initializes the graphics primitive package.  */

std_$call void   gpr_$init ();


/*  GPR_$TERMINATE terminates this package's operation.  */

std_$call void   gpr_$terminate ();
 eject
/*  Set and inquire operations for the display.  */


/*  GPR_$INQ_CONFIG returns the current display configuration.  */

std_$call void   gpr_$inq_config ();


/*  GPR_$SET_COLOR_MAP gives new values for the color map.  */
```

```
std_$call void   gpr_$set_color_map ();


/*  GPR_$INQ_COLOR_MAP returns current values in the color map.  */

std_$call void   gpr_$inq_color_map ();


/*  GPR_$SET_CURSOR_PATTERN loads a cursor pattern.  */

std_$call void   gpr_$set_cursor_pattern ();


/*  GPR_$SET_CURSOR_ACTIVE specifies whether the cursor should be
    on or off.  */

std_$call void   gpr_$set_cursor_active ();


/*  GPR_$SET_CURSOR_POSITION gives the position at which the cursor is
   to be displayed, in the current bitmap.  */

std_$call void   gpr_$set_cursor_position ();


/* GPR_$SET_ORIGIN gives the cursor-relative psoition of its pixel
   which is to be placed at the cursor position.  */

std_$call void   gpr_$set_cursor_origin ();


/*  GPR_$INQ_CURSOR returns information about the cursor.  */

std_$call void   gpr_$inq_cursor ();


/*  GPR_$WAIT_FRAME causes the color display hardware to defer processing
   further requests until the next end-of-frame.  */

std_$call void   gpr_$wait_frame ();
 eject
/*  Bitmap control functions.  */


/*  GPR_$SET_BITMAP establishes the current bitmap for subsequent
    operations.  */
```

```
std_$call void   gpr_$set_bitmap ();


/*  GPR_$INQ_BITMAP returns a descriptor of the current bitmap.  */

std_$call void   gpr_$inq_bitmap ();


/*  GPR_$ALLOCATE_BITMAP allocates a bitmap in main memory.  */

std_$call void   gpr_$allocate_bitmap ();


/*  GPR_$ALLOCATE_BITMAP_NC allocates a bitmap in main memory without
    zeroing it.  */

std_$call void   gpr_$allocate_bitmap_nc ();


/*  GPR_$ALLOCATE_HDM_BITMAP allocates a bitmap in hidden_display memory.  */

std_$call void   gpr_$allocate_hdm_bitmap ();


/*  GPR_$OPEN_BITMAP_FILE obtains access to an external bitmap  */

std_$call void gpr_$open_bitmap_file ();


/*  GPR_$DEALLOCATE_BITMAP deallocates a bitmap allocated by ALLOCATE_BITMAP.  */

std_$call void   gpr_$deallocate_bitmap ();


/*  GPR_$INQ_BITMAP_POINTER returns a pointer to the storage/display memory
   for the given bitmap and the number of words each scan line occupies.
   This information can be used to directly manipulate the bits in the bitmap.  */

std_$call void   gpr_$inq_bitmap_pointer ();


/*  GPR_$INQ_BM_BIT_OFFSET returns the number of bits in the most significant
    part of the first word of each scanline which are not part of the given
    bitmap.  In other words, the offset is the number of bits between a 16-bit
    word boundary and the left edge of the bitmap.
```

```
   Currently, this number can only be nonzero for direct graphics bitmaps.  */

std_$call void   gpr_$inq_bm_bit_offset ();


/*  GPR_$SELECT_COLOR_FRAME selects whether frame 0 (top 1024 lines) or
    frame 1 (bottom 1024 lines) is visible.  Normally frame 0 is visible.  */

std_$call void   gpr_$select_color_frame ();


/*  GPR_$REMAP_COLOR_MEMORY sets the plane of frame 0 of color display memory
    (normally visible) which is mapped at the address returned by
    INQ_BITMAP_POINTER.  */

std_$call void   gpr_$remap_color_memory ();


/*  GPR_$REMAP_COLOR_MEMORY_1 sets the plane of frame 1 of color display
    memory (normally hidden) which is mapped at the address returned by
    INQ_BITMAP_POINTER.  */

std_$call void   gpr_$remap_color_memory_1 ();


/*  GPR_$COLOR_ZOOM sets the zoom scale factors for the color display.  */

std_$call void   gpr_$color_zoom ();


/*  GPR_$ENABLE_DIRECT_ACCESS waits for display hardware to finish current
    operations so that the user can access display memory directly.  */

std_$call void   gpr_$enable_direct_access ();


/*  GPR_$SET_BITMAP_DIMENSIONS changes the size and number of planes
    of the given bitmap.  */

std_$call void   gpr_$set_bitmap_dimensions ();


/*  GPR_$INQ_BITMAP_DIMENSIONS returns the size and number of planes
    of the given bitmap.  */

std_$call void   gpr_$inq_bitmap_dimensions ();
```

```
/*  GPR_$ALLOCATE_ATTRIBUTE_BLOCK allocates an attribute block,
    initialized to default settings.  */

std_$call void   gpr_$allocate_attribute_block ();


/*  GPR_$DEALLOCATE_ATTRIBUTE_BLOCK deallocates an attribute block allocated
    by ALLOCATE_ATTRIBUTE_BLOCK.  */

std_$call void   gpr_$deallocate_attribute_block ();


/*  GPR_$SET_ATTRIBUTE_BLOCK establishes the given attributes as the
     attributes of the current bitmap.  */

std_$call void   gpr_$set_attribute_block ();


/*  GPR_$ATTRIBUTE_BLOCK returns as the function value a descriptor of
    the attributes of the given bitmap.  */

std_$call gpr_$attribute_desc_t gpr_$attribute_block ();
 eject
/*  Set and inquire operations for bitmap attributes.  */


/*  GPR_$SET_attribute sets an attribute in the current bitmap.
    The list of calls follows.  */

std_$call void   gpr_$set_clip_window ();

std_$call void   gpr_$set_clipping_active ();

std_$call void   gpr_$set_text_font ();

std_$call void   gpr_$set_text_path ();

std_$call void   gpr_$set_coordinate_origin ();

std_$call void   gpr_$set_plane_mask ();

std_$call void   gpr_$set_draw_value ();

std_$call void   gpr_$set_text_value ();

std_$call void   gpr_$set_text_background_value ();
```

```
std_$call void   gpr_$set_fill_value ();

std_$call void   gpr_$set_fill_background_value ();

std_$call void   gpr_$set_fill_pattern ();

std_$call void   gpr_$set_raster_op ();

std_$call void   gpr_$set_linestyle ();

std_$call void   gpr_$set_line_pattern ();

std_$call void   gpr_$set_character_width ();

std_$call void   gpr_$set_horizontal_spacing ();

std_$call void   gpr_$set_space_size ();

/*  GPR_$INQ_attributes returns the current settings of a group
    of attributes for the current bitmap.  */

std_$call void   gpr_$inq_constraints ();

std_$call void   gpr_$inq_text ();

std_$call void   gpr_$inq_text_path ();

std_$call void   gpr_$inq_coordinate_origin ();

std_$call void   gpr_$inq_draw_value ();

std_$call void   gpr_$inq_text_values ();

std_$call void   gpr_$inq_fill_value ();

std_$call void   gpr_$inq_fill_background_value ();

std_$call void   gpr_$inq_fill_pattern ();

std_$call void   gpr_$inq_raster_ops ();

std_$call void   gpr_$inq_linestyle ();

std_$call void   gpr_$inq_line_pattern ();

std_$call void   gpr_$inq_character_width ();

std_$call void   gpr_$inq_horizontal_spacing ();

std_$call void   gpr_$inq_space_size ();
 eject
```

```
/*  Drawing operations.  */


/*  GPR_$MOVE sets the CP to the given position.  */

std_$call void   gpr_$move ();


/*  GPR_$INQ_CP returns the current position.  */

std_$call void   gpr_$inq_cp ();


/*  GPR_$LINE draws a line from the CP to the given position
    and sets the CP to the given position.  */

std_$call void   gpr_$line ();


/*  GPR_$POLYLINE does a series of LINEs, starting from the CP.  */

std_$call void   gpr_$polyline ();


/*  GPR_$MULTILINE does a series of alternate MOVEs and LINEs. */

std_$call void   gpr_$multiline ();


/*  GPR_$DRAW_BOX draws a rectangular box  */

std_$call void   gpr_$draw_box ();


/*  GPR_$ARC_3P draws an arc from current point through two points
    p2 and p3. */

std_$call void   gpr_$arc_3p ();


/*  GPR_$CIRCLE draws a circle of radius around point center. */

std_$call void   gpr_$circle ();
```

```
/*  GPR_$CIRCLE_FILLED draws a filled circle of radius around point center. */

std_$call void   gpr_$circle_filled ();


/*  GPR_$SPLINE_CUBIC_P draws a parametric cubic spline through the
    control points. */

std_$call void   gpr_$spline_cubic_p ();


/*  GPR_$SPLINE_CUBIC_X draws a cubic spline as a function of x through
    the control points. */

std_$call void   gpr_$spline_cubic_x ();


/*  GPR_$SPLINE_CUBIC_Y draws a cubic spline as a function of y through
    the control points. */

std_$call void   gpr_$spline_cubic_y ();
 eject
/*  Text operations.  */


/*  GPR_$LOAD_FONT_FILE loads a font contained in a file into an appro-
   priate area (based on the current display mode and configuration).  */

std_$call void   gpr_$load_font_file ();


/*  GPR_$UNLOAD_FONT_FILE unloads a font that has been loaded by
   LOAD_FONT_FILE.  */

std_$call void   gpr_$unload_font_file ();


/*  GPR_$TEXT writes text to a bitmap from the current position ("CP").  */

std_$call void   gpr_$text ();


/*  GPR_$INQ_TEXT_EXTENT returns the x- and y-offsets the given string
   would span if written with TEXT.  */
```

```
std_$call void   gpr_$inq_text_extent ();


/*  GPR_$INQ_TEXT_OFFSET returns the x- and y-offsets that must be added to the
    coordinates of the desired upper left pixel of the string to give the pixel
    from which the TEXT call should be made, and the x_offset of the pixel
    which would be the updated CP.  */

std_$call void   gpr_$inq_text_offset ();


/*  GPR_$REPLICATE_FONT creates and loads a read/write copy of the
    original font. */

std_$call void   gpr_$replicate_font ();
 eject
/*  Data transfer operations.  */


/*  GPR_$CLEAR clears the current bitmap to a given pixel value.  */

std_$call void   gpr_$clear ();


/*  GPR_$READ_PIXELS reads the pixels from the given window of the current
    bitmap and stores them in a pixel array.  */

std_$call void   gpr_$read_pixels ();


/*  GPR_$WRITE_PIXELS writes the pixels from a pixel array into the given
    window of the current bitmap.  */

std_$call void   gpr_$write_pixels ();
 eject
/*  BLT operations.  */


/*  GPR_$PIXEL_BLT moves a rectangle of whole pixels from the source
    bitmap to a position in the current bitmap.  */

std_$call void   gpr_$pixel_blt ();


/*  GPR_$BIT_BLT moves a rectangle of bits from a plane of the source
    bitmap to a position in a plane of the current bitmap.  */
```

```
std_$call void   gpr_$bit_blt ();

/*  GPR_$ADDITIVE_BLT moves a rectangle of bits from a plane of the source
    bitmap to a position in every plane of the current bitmap.  */

std_$call void   gpr_$additive_blt ();
 eject
/*  Fill operations.  */


/*  GPR_$RECTANGLE fills a rectangle in the current bitmap.  */

std_$call void   gpr_$rectangle ();


/*  GPR_$TRAPEZOID fills a trapezoid in the current bitmap.  */

std_$call void   gpr_$trapezoid ();


/*  GPR_$MULTITRAPEZOID fills a list of trapezoids in the current bitmap.  */

std_$call void   gpr_$multitrapezoid ();


/*  GPR_$TRIANGLE fills a triangle in the current bitmap.  */

std_$call void   gpr_$triangle ();


/*  GPR_$START_PGON starts a polygon boundary loop.  */

std_$call void   gpr_$start_pgon ();


/*  GPR_$PGON_POLYLINE defines a series of line segments as part of
    the current polygon boundary loop.  */

std_$call void   gpr_$pgon_polyline ();


/*  GPR_$CLOSE_FILL_PGON closes and fills the currently open polygon.  */

std_$call void   gpr_$close_fill_pgon ();
```

```
/*  GPR_$CLOSE_RETURN_PGON closes the currently open polygon,
    decomposes it, and returns the list of trapezoids.  */

std_$call void   gpr_$close_return_pgon ();
 eject


/* direct graphics */


std_$call void gpr_$set_acq_time_out();


/* GPR_SET_OBSCURED_OPT sets obscured selection attribute for display in
   direct mode */

std_$call void gpr_$set_obscured_opt();


/* GPR_$ACQUIRE_DISPLAY gives the user exclusive access to all display
    operations
   in the acquired window */

std_$call boolean gpr_$acquire_display();


/* GPR_$INQ_VIS_LIST returns list of visible subwindows when a window is
    obscured */

std_$call void gpr_$inq_vis_list();


/* GPR_$FORCE_RELEASE releases the display regardless of how many times it
   was previously acquired */

std_$call void gpr_$force_release();


/* GPR_$RELEASE_DISPLAY will release the display only if acq_rel_cnt is 1 */

std_$call void gpr_$release_display();


/* GPR_$SET_REFRESH_ENTRY provides two std_$call voids which will refresh
    the user's window and refresh hidden display memory */

std_$call void gpr_$set_refresh_entry();


/* GPR_$SET_AUTO_REFRESH tells DM to save bitmap and refresh screen
    when needed */
```

```
std_$call void gpr_$set_auto_refresh();


/* GPR_$EVENT_WAIT hangs until input or time out; returns boolean to indicate
   obscured window */

std_$call boolean gpr_$event_wait();


/* GPR_$COND_EVENT_WAIT returns immediately and reports if any input */

std_$call boolean gpr_$cond_event_wait();


/* GPR_$ENABLE_INPUT enables event type and selected set of keys to be
   recognized by event_wait */

std_$call void gpr_$enable_input();


/* GPR_$DISABLE_INPUT disables event type */

std_$call void gpr_$disable_input();


/* GPR_$GET_EC gets event count */

std_$call void gpr_$get_ec();

/* GPR_$SET_INPUT_SID establishes the stream id for gpr input in frame mode
   if not standard input */

std_$call void gpr_$set_input_sid();

/* GPR_$SET_WINDOW_ID sets the character identifying the current displayed
   bitmap for input identification purposes */

std_$call void gpr_$set_window_id();

/* GPR_$INQ_WINDOW_ID returns the character identifying the current displayed
   bitmap for input identification purposes */

std_$call void gpr_$inq_window_id();
 eject
/* Imaging format calls */

/* GPR_$SET_IMAGING_FORMAT sets the DN600 display format */

std_$call void gpr_$set_imaging_format();

/* GPR_$INQ_IMAGING_FORMAT returns the current DN600 display format */
   std_$call void gpr_$inq_imaging_format();
```

APPENDIX D

INSERT FILE FOR FORTRAN

This appendix contains the insert file for FORTRAN.

```
C---Status code definitions:

      integer*4
     +    gpr_$operation_ok,           gpr_$not_initialized,
     +    gpr_$already_initialized,    gpr_$wrong_display_hardware,
     +    gpr_$illegal_for_frame,      gpr_$must_borrow_display,
     +    gpr_$no_attributes_defined,  gpr_$no_more_space,
     +    gpr_$dimension_too_big,      gpr_$dimension_too_small,
     +    gpr_$bad_bitmap,             gpr_$bad_attribute_block,
     +    gpr_$window_out_of_bounds,   gpr_$source_out_of_bounds,
     +    gpr_$dest_out_of_bounds,     gpr_$invalid_plane,
     +    gpr_$cant_deallocate,        gpr_$coord_out_of_bounds,
     +    gpr_$invalid_color_map,      gpr_$invalid_raster_op,
     +    gpr_$bitmap_is_read_only,    gpr_$internal_error,
     +    gpr_$font_table_full,        gpr_$bad_font_file,
     +    gpr_$invalid_font_id,        gpr_$window_obscured,
     +    gpr_$not_in_direct_mode,     gpr_$not_in_polygon,
     +    gpr_$kbd_not_acq,            gpr_$display_not_acq,
     +    gpr_$illegal_pixel_values,   gpr_$illegal_when_imaging
      integer*4
     +    gpr_$invalid_imaging_format,gpr_$must_release_display,
     +    gpr_$cant_mix_modes,         gpr_$no_input_enabled,
     +    gpr_$duplicate_points,       gpr_$array_not_sorted,
     +    gpr_$character_not_in_font,  gpr_$illegal_fill_pattern,
     +    gpr_$illegal_fill_scale,     gpr_$incorrect_alignment,
     +    gpr_$illegal_text_path,      gpr_$unable_to_rotate_font,
     +    gpr_$font_is_read_only,      gpr_$illegal_pattern_length

      parameter (
     +    gpr_$operation_ok            = 16 00000000,
     +    gpr_$not_initialized         = 16 06010001,
     +    gpr_$already_initialized     = 16 06010002,
     +    gpr_$wrong_display_hardware = 16 06010003,
```

```
     +    gpr_$illegal_for_frame      = 16 06010004,
     +    gpr_$must_borrow_display    = 16 06010005,
     +    gpr_$no_attributes_defined  = 16 06010006,
     +    gpr_$no_more_space          = 16 06010007,
     +    gpr_$dimension_too_big      = 16 06010008,
     +    gpr_$dimension_too_small    = 16 06010009,
     +    gpr_$bad_bitmap             = 16 0601000A,
     +    gpr_$bad_attribute_block    = 16 0601000B,
     +    gpr_$window_out_of_bounds   = 16 0601000C,
     +    gpr_$source_out_of_bounds   = 16 0601000D,
     +    gpr_$dest_out_of_bounds     = 16 0601000E,
     +    gpr_$invalid_plane          = 16 0601000F)
      parameter (
     +    gpr_$cant_deallocate        = 16 06010010,
     +    gpr_$coord_out_of_bounds    = 16 06010011,
     +    gpr_$invalid_color_map      = 16 06010012,
     +    gpr_$invalid_raster_op      = 16 06010013,
     +    gpr_$bitmap_is_read_only    = 16 06010014,
     +    gpr_$internal_error         = 16 06010015,
     +    gpr_$font_table_full        = 16 06010016,
     +    gpr_$bad_font_file          = 16 06010017,
     +    gpr_$invalid_font_id        = 16 06010018,
     +    gpr_$window_obscured        = 16 06010019,
     +    gpr_$not_in_direct_mode     = 16 0601001A,
     +    gpr_$not_in_polygon         = 16 0601001B,
     +    gpr_$kbd_not_acq            = 16 0601001C,
     +    gpr_$display_not_acq        = 16 0601001D,
     +    gpr_$illegal_pixel_values   = 16 0601001E,
     +    gpr_$illegal_when_imaging   = 16 0601001F)
      parameter (
     +    gpr_$invalid_imaging_format = 16 06010020,
     +    gpr_$must_release_display   = 16 06010021,
     +    gpr_$cant_mix_modes         = 16 06010022,
     +    gpr_$no_input_enabled       = 16 06010023,
     +    gpr_$duplicate_points       = 16 06010024,
     +    gpr_$array_not_sorted       = 16 06010025,
     +    gpr_$character_not_in_font  = 16 06010026,
     +    gpr_$illegal_fill_pattern   = 16 06010027,
     +    gpr_$illegal_fill_scale     = 16 06010028,
     +    gpr_$incorrect_alignment    = 16 06010029,
     +    gpr_$illegal_text_path      = 16 0601002A,
     +    gpr_$unable_to_rotate_font  = 16 0601002B,
     +    gpr_$font_is_read_only      = 16 0601002C,
     +    gpr_$illegal_pattern_length = 16 0601002D)


C---Other constant definitions:

      integer*2
     +    gpr_$string_size,
     +    gpr_$max_x_size, gpr_$max_y_size,
     +    gpr_$highest_plane,
     +    gpr_$max_bmf_group,
```

```
     +    gpr_$bmf_major_version,
     +    gpr_$bmf_minor_version
      integer*4
     +    gpr_$black, gpr_$white,
     +    gpr_$red, gpr_$green, gpr_$blue,
     +    gpr_$cyan, gpr_$magenta, gpr_$yellow,
     +    gpr_$transparent, gpr_$background,
     +    gpr_$nil_attribute_desc, gpr_$nil_bitmap_desc

      parameter (
     +    gpr_$string_size = 256,
     +    gpr_$max_x_size = 4096, gpr_$max_y_size = 4096,
     +    gpr_$highest_plane = 7,
     +    gpr_$max_bmf_group = 0,
     +    gpr_$bmf_major_version = 1,
     +    gpr_$bmf_minor_version = 1)
      parameter (
     +    gpr_$black = 0, gpr_$white = 16 FFFFFF,
     +    gpr_$red = 16 FF0000,
     +    gpr_$green = 16 00FF00,
     +    gpr_$blue = 16 0000FF,
     +    gpr_$cyan = 16 00FFFF,
     +    gpr_$magenta = 16 FF00FF,
     +    gpr_$yellow = 16 FFFF00,
     +    gpr_$transparent = -1, gpr_$background = -2,
     +    gpr_$nil_attribute_desc = 0, gpr_$nil_bitmap_desc = 0)


C---the five ways to use this package

      integer*2
     +      gpr_$borrow, gpr_$frame, gpr_$no_display, gpr_$direct,
     +      gpr_$borrow_nc

      parameter (
     +      gpr_$borrow = 0, gpr_$frame = 1, gpr_$no_display = 2,
     +      gpr_$direct = 3, gpr_$borrow_nc = 4)


C---possible display hardware configurations

      integer*2
     +      gpr_$bw_800x1024, gpr_$bw_1024x800,
     +      gpr_$color_1024x1024x4, gpr_$color_1024x1024x8,
     +      gpr_$RESERVEDx4, gpr_$RESERVEDx8

      parameter (
     +      gpr_$bw_800x1024 = 0, gpr_$bw_1024x800 = 1,
     +      gpr_$color_1024x1024x4 = 2, gpr_$color_1024x1024x8 = 3,
     +      gpr_$RESERVEDx4 = 4, gpr_$RESERVEDx8 = 5)


C--imaging vs interactive display formats
```

```
      integer*2
     +      gpr_$interactive,
     +      gpr_$imaging_1024x1024x8,
     +      gpr_$imaging_512x512x24

      parameter (
     +      gpr_$interactive         = 0,
     +      gpr_$imaging_1024x1024x8 = 1,
     +      gpr_$imaging_512x512x24  = 2)

C---directions

      integer*2
     +      gpr_$up, gpr_$down, gpr_$left, gpr_$right

      parameter (
     +      gpr_$up = 0, gpr_$down = 1, gpr_$left = 2, gpr_$right = 3)


C---line styles

      integer*2
     +      gpr_$solid, gpr_$dotted

      parameter (
     +      gpr_$solid = 0, gpr_$dotted = 1)


C---event types

      integer*2
     +      gpr_$keystroke, gpr_$buttons, gpr_$locator,
     +      gpr_$entered_window, gpr_$left_window, gpr_$locator_stop,
     +      gpr_$no_event

      parameter (
     +      gpr_$keystroke = 0, gpr_$buttons = 1, gpr_$locator = 2,
     +      gpr_$entered_window = 3, gpr_$left_window = 4,
     +      gpr_$locator_stop = 5, gpr_$no_event = 6)

C---obscured options for direct graphics

      integer*2
     +      gpr_$ok_if_obs, gpr_$error_if_obs, gpr_$pop_if_obs,
     +      gpr_$block_if_obs

      parameter (
     +      gpr_$ok_if_obs = 0, gpr_$error_if_obs = 1,
     +      gpr_$pop_if_obs = 2, gpr_$block_if_obs = 3)

      integer*2 gpr_$create,gpr_$update, gpr_$write, gpr_$readonly
```

```
      parameter (
     +      gpr_$create = 0,  gpr_$update = 1,
     +      gpr_$write = 2,  gpr_$readonly = 3)


C---function declarations:

      integer*4
     +      gpr_$attribute_block

      logical
     +      gpr_$acquire_display

      logical
     +      gpr_$event_wait

      logical
     +      gpr_$cond_event_wait
```

GLOSSARY

Attribute
Specification of the manner in which a primitive graphic operation is to be performed (for example line type or text value). Each bitmap has a set of attributes.

Bitmap
A three-dimensional array of bits having width, height, and depth. When a bitmap is displayed, it is treated as a two-dimensional array of sets of bits. The color of each displayed pixel is determined by using the set of bits in the corresponding pixel of the frame-buffer bitmap as an index into the color table.

Bit plane
A one-bit-deep layer of a bitmap. On a monochromatic display, displayed bitmaps contain one plane. On a color display, displayed bitmaps may contain more planes, depending on the hardware configuration and the number of bits per pixel.

Borrow display mode
A mode for use of the DOMAIN display whereby a program borrows the entire screen from the Display Manager and performs graphics operations by directly calling the display driver.

Button
A logical input device used to provide a choice from a small set of alternatives. Two physical devices of this type are function keys on a keyboard and selection buttons on a mouse.

Clipping window
A rectangular section of a bitmap outside of which graphics operations do not modify pixels.

Color map
See Color table.

Color table
A set of color table entries, each of which can store one color value. Each color value contains red, blue, and green components. Each entry is accessed by a color table index.

Color table entry
One location in a color table. Each entry stores one color value which can be accessed by a corresponding color table index.

Color table index
An index to a particular color table entry.

Color value
The numeric encoding of a visible color. A color value is stored in a color table entry. Each color value is divided into three fields: the first stores the value of the red component of the color, the second stores the value of the green component of the color, and the third stores the value of the blue component. Each component value is specified as an integer in the range zero to 255, where zero is the absence of the primary color and 255 is the full intensity color.

Core graphics system
A package of graphics functional capabilities designed for building higher-level interactive computer graphics applications programs. Unlike graphics primitives, the Core graphics system allows temporary storage of picture data during execution, with limited segmentation of the pictures. In addition, the Core system uses device-independent coordinates.

Current bitmap
The bitmap on which a program is currently operating.

Current position
In graphics primitives, the starting point of any line drawing and text operations. The current position is initially set at the coordinate position at the top left corner of the bitmap (0,0).

Direct mode
A mode for use of the DOMAIN display whereby the program performs graphics operations in a window borrowed from the Display Manager. Direct mode allows graphics programs to coexist with other activities on the screen, with less Display Manager overhead than frame mode.

Event
An input primitive which is associated with an interrupt from a device such as a keyboard, button, mouse, or touchpad.

Font
One set of alphanumeric and special characters. The font in which text is to be displayed may be specified as an attribute.

Frame
A two-dimensional data structure that holds a picture in a Display Manager pad. This structure is looked at through a Display Manager window. The structure can be larger (or smaller) than the window, and it can be scrolled.

Frame buffer
The digital memory in a raster display unit used to store a bitmap.

Frame mode
A mode for use of the DOMAIN display whereby a program performs graphics operations on a Display Manager pad. In this mode, the user has access to other processes through windows on the display, and can scroll the frame under the Display Manager window. In this mode, unlike direct mode, the Display Manager will refresh the window when appropriate.

Imaging display format
An 8-bit or 24-bit color display format which allows display of an extended color range, but supports only limited graphics primitives operations. In an 8-bit imaging format, 8 bits are used to assign a pixel value (color map index) to each pixel. In a 24-bit imaging format, 24 bits are used to assign a pixel value to each pixel. An 8-bit imaging format allows 256 colors to appear on the screen at one time. A 24-bit imaging format extends the possible color range to 16 million different colors, with 512 x 512 pixels visible at one time.

Initial bitmap
The first bitmap created in a graphics session.

Input device
A device such as a function key, touchpad, or mouse that enables a user to provide input to a program.

Input device number
The identifier of one input device in an input device class.

Interactive display format
A 4-bit or 8-bit color display format which supports all graphics primitives operations. In a 4-bit interactive format, 4 bits are used to assign a pixel value (color map index) to each pixel. In an 8-bit format, 8 bits are used to assign a pixel value to each pixel. A 4-bit format allows sixteen different colors to appear on the screen at one time. An 8-bit format allows 256 colors to appear on the screen at one time.

Keyboard
A logical input device used to provide character or text string input. One physical device of this type is the alphanumeric keyboard.

Line style
An attribute that specifies the style of lines and polylines (for example, solid or dotted).

Locator
A logical input device used to specify one position in coordinate space (for example, a touchpad, data tablet, or mouse).

Logical input device
An abstraction of an input device that provides a particular type of input data. This abstraction corresponds to a group of physical input devices which provide this type of input data.

No-display mode
A mode for use of the DOMAIN system whereby a program creates a bitmap in nondisplayed memory and performs graphic operations there, bypassing the display.

Picture element
A single element of a two-dimensional displayed image or of a two-dimensional location within a bitmap. It is commonly called a pixel.

Pixel
See Picture Element.

Pixel value
The set of bits at a two-dimensional location within a bitmap. A pixel value is used as an index to the color map.

Plane
See Bit plane.

Primitive
The least divisible graphic operation which changes a bitmap (for example, lines, polylines, and text).

Primitive attribute
See Attribute.

RGB color model
A model used to specify color values. It defines red, green, and blue as primary colors. All other colors are combinations of the primaries, including the three secondary colors (cyan, magenta, and yellow).

Scan line
A row of pixels; one horizontal line of a bitmap.

Window
A rectangular area of the visible screen. Parts of the area may be obscured by other windows.

A

B

C

READER'S RESPONSE

We use readers' comments in revising and improving our documents.

Document Title: Programmer's Guide to DOMAIN Graphics Primitives

First Printing: October, 1983
Latest Printing: May, 1984

What is the best feature of this manual? _____

_____

_____

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible.)

_____

_____

_____

_____

What type of user are you?

___ Systems programmer; language _____
___ Applications programmer; language _____
___ Student programmer
___ User with little programming experience

How often do you use your system? _____

Nature of your work on the DOMAIN System:_____

_____

Your name Date

_____

Organization

_____

Street Address

_____

City State Zip/Country

No postage necessary if mailed in the U.S. Fold on dotted lines (see reverse side), tape, and mail.

cut or fold along dotted line

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 100 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER, INC.
16 Elizabeth Drive
Chelmsford, MA 01824

Attn.: Software Documentation

fold

READER'S RESPONSE

We use readers' comments in revising and improving our documents.

Document Title: Programmer's Guide to DOMAIN Graphics Primitives

First Printing: October, 1983
Latest Printing: May, 1984

What is the best feature of this manual? _____

_____

_____

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible.)

_____

_____

_____

_____

What type of user are you?

___ Systems programmer; language _____
___ Applications programmer; language _____
___ Student programmer
___ User with little programming experience

How often do you use your system? _____

Nature of your work on the DOMAIN System:_____

_____
Your name Date

_____
Organization

_____
Street Address

_____
City State Zip/Country

No postage necessary if mailed in the U.S. Fold on dotted lines (see reverse side), tape, and mail.

cut or fold along dotted line

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 100 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER, INC.
16 Elizabeth Drive
Chelmsford, MA 01824

Attn.: Software Documentation

fold

apollo computer inc.
16 Elizabeth Drive, Chelmsford, MA 01824