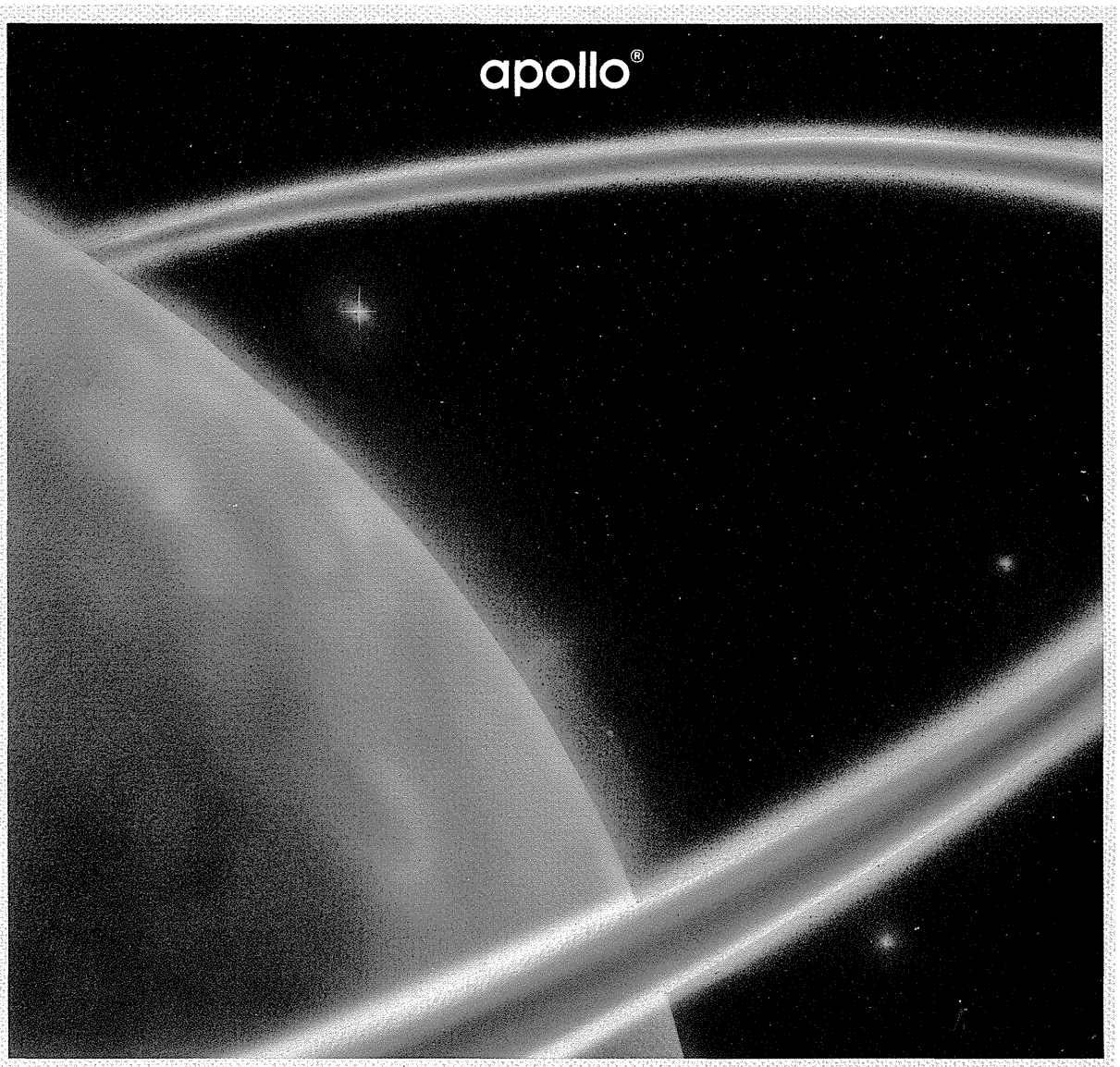


D O M A I N

Order No. 007194
Revision 02

Domain Graphics Primitive Resource Call Reference

Domain Graphics Primitive Resource Call Reference

Order No. 007194
Revision 02

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

Printed in U.S.A.

Latest Printing: June, 1987

This document was produced using the SCRIBE document preparation system. (SCRIBE is a registered trademark of Unilogic, Ltd.)

Preface

The *DOMAIN Graphics Primitive Resource Call Reference* describes the constants, data types, and user-callable routines used by the Domain Graphics Primitive Resource (GPR) system for developing two-dimensional graphics applications.

Audience

This manual is for programmers who use GPR to develop application programs. Users of this manual should have some knowledge of computer graphics and have experience in using the Domain system.

We suggest that you read the task-oriented handbook *Programming with Domain Graphics Primitives* before using this reference manual.

Organization of this Manual

This manual contains three chapters:

Chapter 1	Presents the constants and data types used by GPR.
Chapter 2	Describes each GPR routine. We've organized the routines in alphabetical order.
Chapter 3	Lists all the GPR runtime error messages.

Additional Reading

Use this reference as a companion to the *Programming With Domain Graphics Primitives* manual (005808).

The *Domain 3D Graphics Metafile Resource Call Reference* manual (005812) describes the constants, data types, and user-callable routines used by the Domain 3D Graphics Metafile Resource (3D GMR) system for developing three-dimensional graphics applications.

The *Programming With Domain 3D Graphics Metafile Resource* manual (005807) describes how to write programs that use the Domain 3D Graphics Metafile Resource.

The *Domain 2D Graphics Metafile Resource Call Reference* manual (009793) describes the constants, data types, and user-callable routines used by the Domain 2D Graphics Metafile Resource (GMR) system for developing two-dimensional graphics applications.

The *Programming With Domain 2D Graphics Metafile Resource* manual (005097) describes how to write graphics programs using the Domain 2D Graphics Metafile Resource (GMR) system.

The *Programming With General System Calls* manual (005506) describes how to write programs that use standard Domain systems calls.

The *Domain Language Level Debugger Reference* (001525) describes the high-level language debugger.

The *Programming With Graphics Service Routines* (009797) manual describes how to write programs that use Graphics Service Routines.

The *Domain Graphics Instruction Set* (009791) manual describes the instruction set used by the Graphics Service Routines.

For language-specific information, see the *Domain FORTRAN Language Reference* (000530), the *Domain Pascal Language Reference* (000792), or the *Domain C Language Reference* (002093).

Documentation Conventions

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

lowercase	Lowercase words or characters in format descriptions represent values that you must supply.
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.

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) Shell command. You can also view the same description on-line by typing:

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

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

Summary of Technical Changes

This release of GPR (SR9.6) introduces the following new calls:

- gpr_$arc_c2p draws an arc from the current position to the point where the arc intersects a user defined ray.
- gpr_$set_draw_width sets the draw width in pixels for all line and curve primitives.
- gpr_$inq_draw_width returns the draw width for all line and curve primitives.
- gpr_$set_draw_pattern specifies the draw pattern to use in draw routines.
- gpr_$inq_draw_pattern returns the draw pattern.

- gpr_$set_plane_mask_32 is just like the gpr_$set_plane_mask call except that it sets a mask on 32 planes instead of only 8 planes.

- GPR now supports double buffering on the DN590 (with certain restrictions). To support double buffering, we've added four new GPR calls. Use the new gpr_$allocate_buffer call to create a buffer bitmap having the same size and attributes as the primary bitmap. Use the new gpr_$deallocate_buffer call to remove the buffer bitmap. Use the new gpr_$select_display_buffer call to control which of the two bitmaps (the original bitmap or the buffer bitmap) is displayed. Use the new gpr_$inq_visible_buffer_call to learn which of the two bitmap is currently displayed.

In addition, we've made the following additional enhancements to GPR:

- The data type gpr_$display_mode_t supports the following three additional display modes: gpr_$direct_rgb, gpr_$borrow_rgb, and gpr_$borrow_rgb_nc. Use these new modes to initialize true-color (24-plane) programs.

- We've added a new data type named gpr_$rgb_plane_t which obsoletes the gpr_$plane_t data type. In Pascal, gpr_$rgb_plane_t is a subrange of integers from 0 to 31. The GPR calls that used gpr_$plane_t parameters (e.g., gpr_$init, gpr_$set_bitmap_dimensions, gpr_$allocate_bitmap) now take gpr_$rgb_plane_t parameters instead. Existing programs will not break as a result of this change. As of SR9.6, you can declare a hi_plane_id of 0 to 31 for a main memory bitmap or 0 to 23 for a display memory bitmap.

- We've added new fields to the gpr_$disp_char_t data type, and enhanced the gpr_$inq_disp_characteristics call.

- GPR now supports pixel-oriented bitmaps. We have not created any new calls to support pixel-oriented bitmaps. Instead, we enhanced the gpr_$open_bitmap_file call. In a pixel-oriented bitmap, a byte contains eight planes of one pixel. By comparison, in plane-oriented bitmaps, a byte contains one plane of eight different pixels.

Contents

Chapter 1 Constants and Data Types **1-1**

1.1. Simulating Enumerated Variables in FORTRAN 1-1
1.2. Simulating Set Variables in C and FORTRAN 1-1
1.3. Simulating Record Types in FORTRAN 1-2
1.4. Simulating Pointer Types in FORTRAN 1-2

Chapter 2 GPR Routines **2-1**

Chapter 3 GPR Errors **3-1**

Index **Index-1**

Chapter 1
Constants and Data Types

This chapter describes the constants and data types used by the Graphics Primitive Resource package (hereafter referred to as GPR).

We've listed all the GPR data types in alphabetical order. If you are writing GPR programs in Pascal, then all GPR data types are predefined in the /sys/ins/gpr.ins.pas file. If you are writing GPR programs in C, then most of the GPR data types are predefined in the /sys/ins/gpr.ins.c file. However, the GPR set data types are not predefined; C GPR programmers must learn to simulate these data types. If you are writing GPR programs in FORTRAN, then none of the GPR data types are predefined; you must learn how to simulate all GPR data types.

The individual descriptions of each GPR data type explain how to simulate the type in FORTRAN (or C if necessary). In addition, you may find the following general notes to be useful.

1.1. Simulating Enumerated Variables in FORTRAN

Pascal and C both support enumerated variables, but FORTRAN does not. However, the DOMAIN system stores enumerated Pascal variables and short enum C variables the same way it stores FORTRAN INTEGER*2 variables. Therefore, we've simulated enumerated constants in the /sys/ins/gpr.ins.ftn insert file by defining INTEGER*2 parameters.

If a GPR call requires an enumerated variable, you should declare the variable in your FORTRAN program as an INTEGER*2. To set the variable's value, you merely specify one of the listed choices and the compiler will convert it to the necessary internal representation.

For example, to simulate a gpr_$display_mode_t variable, you can make the following declaration:

```
INTEGER*2  my_display_mode_variable
```

You can set this variable to any one of the listed choices; for example:

```
my_display_mode_variable = gpr_$borrow
```

1.2. Simulating Set Variables in C and FORTRAN

Pascal supports set variables, but C and FORTRAN do not. However, C and FORTRAN programmers can simulate Pascal set variables. If the base type of the Pascal set contains 16 or fewer members, you can simulate the set by declaring a 2-byte integer. If the base type of the set contains 17-32 members, you can simulate the set by declaring a 4-byte integer. If the base type of the set contains more than 32 members, then you must use the special set emulation functions to simulate the set. The descriptions in this chapter will tell you which simulation method is appropriate for a specific data type. For full details on set simulation, see the *Programming With General Systems Calls* manual.

1.3. Simulating Record Types in FORTRAN

Pascal supports record types which are identical to C's structure types. However, FORTRAN does not support such a structure. Nevertheless, you can usually use a FORTRAN array variable to simulate a Pascal record/C structure variable. Nearly all GPR record types can be simulated in FORTRAN by declaring an array of INTEGER*2.

For example, consider our description of the gpr_$offset_t record type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
x_size	integer16/short int	1
y_size	integer16/short int	2

FORTRAN programmers can easily simulate this type by declaring a variable as a 2-element array of INTEGER*2's. After declaring this variable, you can then access the x_size field by accessing array element 1 and the y_size field through array element 2.

1.4. Simulating Pointer Types in FORTRAN

Both Pascal and C support pointer types, but standard FORTRAN does not. However, you can emulate a pointer variable in Domain FORTRAN by declaring an INTEGER*4 variable and then writing addresses into it with the IADDR function. (See the Domain FORTRAN Language Reference) manual for details.

CONSTANTS

Mnemonic	Value	Explanation
gpr_$background	-2	Pixel value for window background.
gpr_$black	0	Color value for black.
gpr_$blue	16#0000FF	Color value for blue.
gpr_$bmf_major_version	1	Major identifier for a bitmap file.
gpr_$bmf_minor_version	1	Minor identifier for a bitmap file.
gpr_$cyan	16#00FFFF	Color value for cyan (blue + green).
gpr_$default_list_size	10	Default number of elements in certain arrays.
gpr_$green	16#00FF00	Color value for green.
gpr_$highest_plane	7	Max. plane number in a bitmap.
gpr_$highest_rgb_plane	31	Max. plane number in a true-color bitmap.
gpr_$magenta	16#FF00FF	Color value for magenta (red + blue).
gpr_$max_bmf_group bitmap.	0	Max. number of groups in an external
gpr_$max_x_size	8192	Max. bits in bitmap x dimension.
gpr_$max_y_size	8192	Max. bits in bitmap y dimension.
gpr_$nil_attribute_desc	0	Descriptor of nonexistent attributes.
gpr_$nil_bitmap_desc	0	Descriptor of a nonexistent bitmap.
gpr_$red	16#FF0000	Color value for red.
gpr_$rop_zeros	0	
gpr_$rop_src_and_dst	1	
gpr_$rop_src_and_not_dst	2	
gpr_$rop_src	3	
gpr_$rop_not_src_and_dst	4	
gpr_$rop_dst	5	
gpr_$rop_src_xor_dst	6	
gpr_$rop_src_or_dst	7	
gpr_$rop_not_src_and_not_dst	8	
gpr_$rop_src_equiv_ds	9	
gpr_$rop_not_dst	10	
gpr_$rop_src_or_not_dst	11	

gpr_$rop_not_src	12	
gpr_$rop_not_src_or_dst	13	
gpr_$rop_not_src_or_not_ds	14	
gpr_$rop_ones	15	
gpr_$string_size	256	Number of chars in a gpr string.
gpr_$transparent	-1	Pixel value for transparent (no change).
gpr_$white	16#FFFFFF	Color value for white.
gpr_$yellow	16#FFFF00	Color value for yellow (red + green).

DATA TYPES

gpr_$accelerator_type_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type holds a unique number corresponding to the graphics accelerator processor type. In Pascal, FORTRAN, or C, gpr_$accelerator_type_t must be equal to one of the following predefined values:

gpr_$accel_none
None or not applicable.

gpr_$accel_1
3DGA.

gpr_$access_allocation_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This parameter holds the legal pixel cell size, in bits, which are available to a program making direct read or write access to the refresh buffer. In Pascal, FORTRAN, or C, gpr_$access_allocation_t must be equal to one of the following predefined values:

gpr_$alloc_1
One bit per pixel cell.

gpr_$alloc_2
Two bits per pixel cell.

gpr_$alloc_4
Four bits per pixel cell.

gpr_$alloc_8
One byte per pixel cell.

gpr_$alloc_16
Two bytes per pixel cell.

gpr_$alloc_32
Four bytes per pixel cell.

gpr_$access_mode_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type defines the method of accessing an external bitmap. In Pascal, FORTRAN, or C, gpr_$access_mode_t must be equal to one of the following predefined values:

gpr_$create
Create a file on disk.

gpr_$update
Update a file on disk.

gpr_$write
Write to a file on disk.

gpr_$readonly
Read a file on disk.

gpr_$access_set_t

This is a predefined set of gpr_$access_allocation_t type in Pascal. C and FORTRAN do not support set types, but you can simulate this type by declaring a short int variable in C or an INTEGER*2 variable in FORTRAN. This set has 6 members. This parameter gives the possible legal pixel cell sizes, in bits, which are available to a program making direct read or write access to the refresh buffer. Currently, the only supported pixel cell size is one bit. This means that the refresh buffers can only be accessed by plane. In the future, other pixel cell sizes may be supported.

gpr_$arc_direction_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the direction to draw an arc. In Pascal, FORTRAN, or C, gpr_$arc_direction_t must be equal to one of the following predefined values:

gpr_$arc_ccw
Draw arc counter-clockwise.

gpr_$arc_cw
Draw arc clockwise.

gpr_$arc_option_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the action to take when the end points of the arc are coincident. In Pascal, FORTRAN, or C, gpr_$arc_option_t must be equal to one of the following predefined values:

gpr_$arc_draw_none
Draw nothing.

gpr_$arc_draw_full
Draw a full circle.

gpr_$attribute_desc_t	This is a predefined unsigned 32-bit integer type in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*4 variable. This type identifies an attribute block.
gpr_$bitmap_desc_t	This is a predefined unsigned 32-bit integer type in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*4 variable. This type identifies a bitmap.

gpr_$bmf_group_header_t

This is a predefined record type in Pascal and a predefined structure type in C. This type holds the group header description for an external bitmap. FORTRAN does not support record/structure types, but you can simulate this type with the following declarations:

```
INTEGER*2 grouparray(8)
CHARACTER*1 storage_offset
INTEGER*2 n_sects,pixel_size
INTEGER*2 allocated_size,bytes_per_line
INTEGER*4 bytes_per_sect,a_pointer
EQUIVALENCE (grouparray(1), n_sects)
EQUIVALENCE (grouparray(2), pixel_size)
EQUIVALENCE (grouparray(3), allocated_size)
EQUIVALENCE (grouparray(4), bytes_per_line)
EQUIVALENCE (grouparray(5), bytes_per_sect)
EQUIVALENCE (grouparray(7), a_pointer)
POINTER /a_pointer/ storage_offset
```

The diagram below illustrates the gpr_$bmf_group_header_t record/structure type:

Name of Field	Data type of Field in Pascal/C
n_sects	integer16/short int
pixel_size	integer16/short int
allocated_size	integer16/short int
bytes_per_line	integer16/short int
bytes_per_sect	integer32/long int
storage_offset	univ_ptr/*char

Description of Each Field:

n_sects
The number of sections in a group.

pixel_size
The number of bits per pixel in each section of a group.

allocated_size
The number of bits that the system uses to store the value of one pixel.

bytes_per_line
The number of bytes in one row of a bitmap.

bytes_per_sect
The value of bytes_per_line multiplied by the height of the bitmap. This value must be rounded up to a page boundary, or for small bitmaps rounded up to the next largest binary submultiple of a page.

storage_offset
A pointer to the group storage area.

gpr_$bmf_group_header_array_t
: An array of up to gpr_$max_bmf_group elements. Each element of the array has the data type gpr_$bmf_group_header_t.

gpr_$color_t
: This is a predefined unsigned 32-bit integer type in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*4 variable. This type holds the red, green, and blue intensity values for one color.

gpr_$color_vector_t
: This is a predefined 256-element array of gpr_$color_t in Pascal and C. You can simulate this type in FORTRAN by declaring a 256-element array of INTEGER*4 variable. This type stores an array of color values. You can use this data type to store the values that comprise your color map.

gpr_$controller_type_t
: This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies a unique number corresponding to the display controller type. In Pascal, FORTRAN, or C, gpr_$controller_type_t must be equal to one of the following predefined values:

  gpr_$ctl_none
  None or not applicable.

  gpr_$ctl_mono_1
  DN100/400/420/460.

  gpr_$ctl_mono_2
  DN300/320/330.

  gpr_$ctl_color_1
  DN600/660/550/560.

  gpr_$ctl_color_2
  DN580/580-T/590/590-T.

  gpr_$ctl_color_3
  DN570/570A/570-T.

  gpr_$ctl_color_4
  DN3000 1024x800 color.

  gpr_$ctl_mono_4
  DN3000 mono.

  gpr_$ctl_color_5
  DN3000 1280x1024 color.

gpr_$coordinate_array_t

This is a predefined 16384-element array of gpr_$coordinate_t in Pascal and C. You can simulate this type in FORTRAN by declaring a 16384-element array of INTEGER*2 variable. This type specifies several coordinates. Generally, x coordinates are passed in one array and y coordinates are passed in another array.

gpr_$coordinate_t

This is a predefined unsigned 16-bit integer in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*2 variable. This type specifies one coordinate in a bitmap.

gpr_$decomp_technique_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies a decomposition technique. In Pascal, FORTRAN, or C, gpr_$decomp_technique_t must be equal to one of the following predefined values:

gpr_$fast_traps
Decomposes polygons into trapezoids using integer arithmetic.

gpr_$precise_traps
Decomposes polygons into trapezoids using double integer arithmetic.

gpr_$non_overlapping_tris
Decomposes polygons into nonoverlapping triangles.

gpr_$render_exact
Renders polygons directly without decomposing them into simpler polygons.

gpr_$direction_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the direction of movement from one text character position to another in a bitmap. In Pascal, FORTRAN, or C, gpr_$direction_t must be equal to one of the following predefined values:

gpr_$up
Write the text from bottom to top.

gpr_$down
Write the text from top to bottom.

gpr_$left
Write the text from right to left.

gpr_$right
Write the text from left to right.

gpr_$disp_char_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 31-element array of INTEGER*2. (Note that you can optionally declare a smaller array.) The gpr_$disp_char_t type stores display characteristics. The diagram below illustrates the gpr_$disp_char_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
controller_type	gpr_$controller_type_t	1
accelerator_type	gpr_$accelerator_type_t	2
x_window_origin	integer16/short int	3
y_window_origin	integer16/short int	4
x_window_size	integer16/short int	5
y_window_size	integer16/short int	6
x_visible_size	integer16/short int	7
y_visible_size	integer16/short int	8
x_extension_size	integer16/short int	9
y_extension_size	integer16/short int	10
x_total_size	integer16/short int	11
y_total_size	integer16/short int	12
x_pixels_per_cm	integer16/short int	13
y_pixels_per_cm	integer16/short int	14
n_planes	integer16/short int	15
n_buffers	integer16/short int	16
delta_x_per_buffer	integer16/short int	17
delta_y_per_buffer	integer16/short int	18
delta_planes_per_buffer	integer16/short int	19
mem_overlaps	gpr_$overlap_set_t	20
x_zoom_max	integer16/short int	21
y_zoom_min	integer16/short int	22
video_refresh_rate	integer16/short int	23
n_primaries	integer16/short int	24
lut_width_per_primary	integer16/short int	25
avail_formats	gpr_$format_set_t	26
avail_access	gpr_$access_set_t	27
access_address_space	integer16/short int	28
invert	gpr_disp_invert_t	29
num_lookup_tables	integer16/short int	30
rgb_color	gpr_$rgb_modes_set_t	31

Description of Each Field:

controller_type
The type of graphics hardware controller in gpr_$controller_type_t format. For gpr_$no_display mode, gpr_$ctl_none is returned.

accelerator_type
The type of graphics hardware processing accelerator for the node in gpr_$accelerator_type_t format. For gpr_$no_display mode, gpr_$accel_none is returned.

x_window_origin
X origin of the frame or window in frame and direct mode respectively. For borrow mode and no-display mode the origin is (0,0).

y_window_origin
Y origin of the frame or window in frame and direct mode respectively. For borrow mode and no-display mode the origin is (0,0).

x_window_size
X dimension of the frame or window in frame and direct mode respectively. For borrow mode this is the x dimension of the screen. For no-display mode this is the x dimension of the maximum legal bitmap.

y_window_size
Y dimension of the frame or window in frame and direct mode respectively. For borrow mode this is the x dimension of the screen. For no-display mode this is the y dimension of the maximum legal bitmap.

x_visible_size
X dimension of the visible area of the screen for frame, direct, and borrow modes. For no-display mode this is the x dimension of the maximum legal bitmap size.

y_visible_size
Y dimension of the visible area of the screen for frame, direct, and borrow modes. For no-display mode this is the y dimension of the maximum legal bitmap size.

x_extension_size
The maximum x dimension of the bitmap after having been extended by gpr_$set_bitmap_dimensions. For frame, direct and no-display modes, this size is the same as x_visible_size. For borrow-mode, this size may be bigger if the device has more display memory past the edges of the visible area.

y_extension_size
The maximum y dimension of the bitmap after having been extended by gpr_$set_bitmap_dimensions. For frame, direct and no-display modes, this size is the same as y_visible_size. For borrow-mode, this size may be bigger if the device has more display memory past the edges of the visible area.

x_total_size
X dimension of total bitmap memory. In particular, this is the number of addressable pixel positions, in a linear pixel addressing space, between the first pixel of a scan line and the first pixel of the next scan line. This value may be larger than x_extension_size. For no-display mode this value is the x dimension of the maximum legal bitmap.

y_total_size
Y dimension of total bitmap memory. This value may be larger than y_extension_size. For no-display mode this value is the y dimension of the maximum legal bitmap.

x_pixels_per_cm
The number of physical pixels per centimeter on the screen in the x dimension. For no-display mode, this value is set to zero.

y_pixels_per_cm
The number of physical pixels per centimeter on the screen in the y dimension. For no-display mode, this value is set to zero.

n_planes
The maximum number of planes of bitmap memory available on the device. For no-display mode, this parameter is the maximum legal bitmap depth.

n_buffers
The number of displayable refresh buffers available on the device, in borrow mode. In frame, direct, and no-display modes, this parameter is set to one.

delta_x_per_buffer
The "distance" in x, in pixel addresses between refresh buffers on a device with more than one buffer, in borrow mode. For frame, direct, and no-display modes, and for devices with only one buffer, this parameter is set to zero.

delta_y_per_buffer
The "distance" in y, in pixel addresses between refresh buffers on a device with more than one buffer, in borrow mode. For frame, direct, and no-display modes, and for devices with only one buffer, this parameter is set to zero.

delta_planes_per_buffer
This parameter gives the "distance" in pixel depth between refresh buffers on a device with more than one buffer, in borrow mode. Currently no such device capability is supported, but it may be in the future. For frame, direct, and no-display modes, and for devices with only one buffer, this parameter is set to zero.

mem_overlaps
The kinds of overlap situations that can exist in refresh buffer memory in gpr_$overlap_set_t format.

x_zoom_max
The maximum pixel-replication zoom factor for x on a device in borrow mode. For frame, direct, and no-display modes, and for devices that do not support pixel-replication zoom, these parameters are set to 1.

y_zoom_max
The maximum pixel-replication zoom factor for y on a device in borrow mode. For frame, direct, and no-display modes, and for devices that do not support pixel-replication zoom, these parameters are set to 1.

video_refresh_rate
The refresh rate of the screen in Hertz. For no-display mode, this value is set to zero.

n_primaries
The number of independent primary colors supported by the video for the device. For color devices, this value is three; for monochrome devices it is one. For no-display mode, this value is set to zero.

lut_width_per_primary
The value gives the number of bits of precision available in each column of a video lookup table (color map) for representing the intensity of a primary color in an overall color value. If a primary color can only be on or off, this value is one. If it can have 16 intensities, this value will be four. If it can have 256 intensities, this value will be eight. For no-display mode, this parameter is set to zero.

avail_formats
The set of available interactive or imaging formats available on the device in gpr_$format_set_t format.

avail_access
The set of legal pixel cell sizes in gpr_$access_set_t format.

access_address_space
This parameter gives the amount of address space available for making direct access to the refresh buffer of the device, in units of 1K-byte pages. For example, if the address space is of a size sufficient to cover 1024 scan lines, each of 1024 bits, its extent will be 128K bytes, thus the value of this parameter will be 128.

invert
This parameter is intended for monochromatic devices. It indicates how the display manager's INV command is implemented on the device in gpr_$disp_invert_t format.

num_lookup_tables
This parameter returns the number of lookup tables available on this node. All current Apollo nodes support only 1 lookup table.

rgb_color
This parameter tells you what kinds of lookup modes are supported by the machine, in gpr_$rgb_modes_set_t format.

gpr_$disp_invert_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This data type holds a value which tells you how the Display Manager's INV command is implemented on the target monochromatic node. In Pascal, FORTRAN, or C, gpr_$disp_invert_t must be equal to one of the following predefined values:

gpr_$no_invert
The display is not a monochromatic display or there is no display.

gpr_$invert_simulate
Color map is simulated in software.

gpr_$invert_hardware
Color map is implemented in hardware.

gpr_$display_config_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the hardware configuration. In Pascal, FORTRAN, or C, gpr_$display_config_t must be equal to one of the following predefined values:

gpr_$bw_800x1024
A portrait black and white display.

gpr_$bw_1024x800
A landscape black and white display.

gpr_$color_1024x1024x4
A 1024x1024 four-plane color display.

gpr_$color_1024x1024x8
A 1024x1024 eight-plane color display.

gpr_$color_1024x800x4
A 1024x800 four-plane color display.

gpr_$color_1024x800x8
A 1024x800 eight-plane color display.

gpr_$color_1280x1024x8
A 1280x1024 eight-plane color display.

gpr_$color1_1024x800x8
A 1024x800 eight-plane color display.

gpr_$color2_1024x800x4
A 1024x800 four-plane color display.

gpr_$bw_1280x1024
A 1280x1024 black and white display.

gpr_$color2_1024x800x8
A 1024x800 eight-plane color display.

gpr_$display_invert_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies how the target monochromatic node implements the color map. In Pascal, FORTRAN, or C, gpr_$display_invert_t must be equal to one of the following predefined values:

gpr_$no_invert
Not applicable, that is, the target node is a color monitor or is not a display.

gpr_$invert_simulate
The color map is simulated in software.

gpr_$invert_hardware
The color map is in hardware.

gpr_$display_mode_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the mode of operation. In Pascal, FORTRAN, or C, gpr_$display_mode_t must be equal to one of the following predefined values:

gpr_$borrow
Uses the entire screen for a pseudo-color program.

gpr_$frame
Uses a frame of the Display Manager for a pseudo-color program.

gpr_$no_display
Uses a main-memory bitmap.

gpr_$direct
Uses a display-manager window for a pseudo-color program.

gpr_$borrow_nc
Uses the entire screen for a pseudo-color program but does not clear the bitmap.

gpr_$direct_rgb
(New mode.) Uses a display-manager window for a true-color program.

gpr_$borrow_rgb
(New mode.) Uses the entire screen for a true-color program.

gpr_$borrow_rgb_nc
(New mode.) Uses the entire screen for a true-color program but does not clear the bitmap.

gpr_$double_buffer_option_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type tells the system what to do to the designated bitmap in a double-buffering application. In Pascal, FORTRAN, or C, gpr_$double_buffer_option_t must be equal to one of the following predefined values:

gpr_$undisturbed_buffer
Do nothing to the specified bitmap.

gpr_$clear_buffer
Clear the option buffer to the color specified by the option_value parameter.

gpr_$copy_buffer
Copy the display_desc bitmap to the option_desc bitmap.

gpr_$ec_key_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies which eventcount to obtain. Currently, there is only one possible value for gpr_$ec_key_t, and it is called gpr_$input_ec.

gpr_$event_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This data type specifies the type of input event. In Pascal, FORTRAN, or C, gpr_$event_t must be equal to one of the following predefined values:

gpr_$keystroke
When you type a keyboard character.

gpr_$buttons
When you press a button on the mouse or bitpad puck.

gpr_$locator
When you move the mouse or bitpad puck, or use the touchpad.

gpr_$locator_update
Only the most recent location when you move the mouse or bitpad puck, or use the touchpad.

gpr_$entered_window
When the cursor enters a window in which the GPR bitmap resides. Direct mode is required.

gpr_$left_window
When the cursor leaves a window in which the GPR bitmap resides. Direct mode is required.

gpr_$locator_stop
When you stop moving the mouse or bitpad puck, or stop using the touchpad.

gpr_$no_event
When you do not enter any events.

gpr_$format_set_t

This is a predefined set of gpr_$imaging_format_t type in Pascal. C and FORTRAN do not support set types, but you can simulate this type by declaring a short int variable in C or an INTEGER*2 variable in FORTRAN. This set has 3 members. This data type specifies the set of interactive or imaging formats available on the device.

gpr_$horiz_seg_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 3-element array of INTEGER*2. This type defines the left-hand x-coordinate, right-hand x-coordinate, and y-position of either the base or roof of a trapezoid. The diagram below illustrates the gpr_$horiz_seg_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
x_coord_l	integer16/short int	1
x_coord_r	integer16/short int	2
y_coord	integer16/short int	3

Description of Each Field:

x_coord_l
The left-hand x-coordinate of the line.

x_coord_r
The right-hand x-coordinate of the line.

y_coord
The y-coordinate of the line.

gpr_$imaging_format_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies an imaging or interactive display format. In Pascal, FORTRAN, or C, gpr_$imaging_format_t must be equal to one of the following predefined values:

gpr_$interactive
Specifies interactive format.

gpr_$imaging_1024x1024x8
Specifies 8-bit imaging format.

gpr_$imaging_512x512x24
Specifies 24-bit imaging format.

gpr_$keyset_t

This is a predefined set of char type in Pascal. Neither C nor FORTRAN supports sets, so you must use the set emulation calls to add elements to or remove elements from the "set". You can reserve the appropriate amount of space in C by declaring a variable as gpr_$keyset_t. FORTRAN programs can reserve the appropriate amount of space by declaring a variable as an 8-element array of INTEGER*4. The gpr_$keyset_t type specifies the set of characters that make up a keyset associated with the graphics input event types gpr_$keystroke and gpr_$buttons.

gpr_$line_pattern_t

This is a predefined 4-element array of 2-byte integers in Pascal and C. You can simulate this type in FORTRAN by declaring a 4-element array of INTEGER*2 variable. This type specifies the line-pattern to use for line-drawing operations.

gpr_$linestyle_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the linestyle for line-drawing operations. In Pascal, FORTRAN, or C, gpr_$linestyle_t must be equal to one of the following predefined values:

gpr_$solid
Draw solid lines.

gpr_$dotted
Draw dotted lines.

gpr_$mask_t

This is a predefined unsigned 16-bit integer in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*2 variable. This type specifies a set of planes to be used in a 16-bit plane mask.

gpr_$mask_32_t

This is a predefined unsigned 32-bit integer type in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*4 variable. This type specifies a set of planes to be used in a 32-bit plane mask.

gpr_$memory_overlap_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the kinds of memory overlaps existing between different classes of buffer memory. In Pascal, FORTRAN, or C, gpr_$memory_overlap_t must be equal to one of the following predefined values:

gpr_$hdm_with_bitm_ext
Hidden display memory (HDM), used for loaded text fonts and HDM bitmaps, overlaps with the area into which a bitmap can be extended by use of the gpr_$set_bitmap_dimensions call.

gpr_$hdm_with_buffers
HDM overlaps with extra displayable refresh buffers.

gpr_$bitm_ext_with_buffers
The bitmap extension area overlaps with displayable refresh buffers.

gpr_$obscured_opt_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the action to be taken when a window is obscured. In Pascal, FORTRAN, or C, gpr_$obscured_opt_t must be equal to one of the following predefined values:

gpr_$ok_if_obs
Acquire the display even though the window is obscured.

gpr_$input_ok_if_obs
Acquire the display and allow input into the window even though the window is obscured.

gpr_$error_if_obs
Do not acquire the display; return an error message.

gpr_$pop_if_obs
Pop the window if it is obscured.

gpr_$block_if_obs
Do not acquire the display until the window is popped.

gpr_$offset_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 2-element array of INTEGER*2. This type specifies the width and height of a window. The diagram below illustrates the gpr_$offset_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
x_size	integer16/short int	1
y_size	integer16/short int	2

Description of Each Field:

x_size
The width of the window in pixels.

y_size
The height of the window in pixels.

gpr_$overlap_set_t

This is a predefined set of gpr_$memory_overlap_t type in Pascal. C and FORTRAN do not support set types, but you can simulate this type by declaring a short int variable in C or an INTEGER*2 variable in FORTRAN. This set has 3 members. This type specifies a set of overlaps between different classes of buffer memory. Sometimes a device comes with extra refresh buffer memory beyond what is used to hold the screen image. There are several recognized purposes for particular parts of such memory, and sometimes some memory locations may be available for more than one purpose. If so, the program using this memory will have to take care not to use the same memory for two different purposes at the same time. In order to decide whether this is a possibility, the program can inspect this parameter. For frame, direct and no-display modes, this parameter is set to the null set.

gpr_$pixel_array_t

This is a predefined 131073-element array of 4-byte integers in Pascal and C. You can simulate this type in FORTRAN by declaring a 131073-element array of INTEGER*4 variable. This type stores multiple pixel values.

gpr_$pixel_value_t

This is a predefined unsigned 32-bit integer type in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*4 variable. This type defines an index into a color map to identify the color of an individual pixel.

gpr_$plane_t

This is a predefined unsigned 16-bit integer in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*2 variable. This type specifies the number of planes in a bitmap; this value will fall between 0 and 7 inclusive.

gpr_$position_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 2-element array of INTEGER*2. This type specifies the x and y coordinates of a point in a bitmap. The diagram below illustrates the gpr_$position_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
x_coord	integer16/short int	1
y_coord	integer16/short int	2

Description of Each Field:

x_coord
The x-coordinate of the point in the bitmap.

y_coord
The y-coordinate of the point in the bitmap.

gpr_$raster_op_array_t

This is a predefined 8-element array of gpr_$raster_op_t in Pascal and C. You can simulate this type in FORTRAN by declaring a 8-element array of INTEGER*2 variable. This type stores multiple raster operation opcodes.

gpr_$raster_op_t

This is a predefined unsigned 16-bit integer in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*2 variable. This type specifies raster operation opcodes.

gpr_$rgb_modes_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the kinds of red, green, and blue lookup supported by the target hardware. In Pascal, FORTRAN, or C, gpr_$rgb_modes_t must be equal to one of the following predefined values:

gpr_$rgb_none
All current nodes except the DN590 will return this value. It indicates that only pseudo-color lookup is available on the node. That is, the node views a pixel's color as an index into the color table.

gpr_$rgb_24
The DN590 node will return this value. It indicates that 24-bit true-color lookup is available on the node. That is, the node can view a pixel's color as three separate indices into the color table.

gpr_$rgb_modes_set_t

This is a predefined set of gpr_$rgb_modes_t type in Pascal. C and FORTRAN do not support set types, but you can simulate this type by declaring a short int variable in C or an INTEGER*2 variable in FORTRAN. This set has 2 members.

gpr_$rgb_plane_t

This is a predefined unsigned 32-bit integer type in Pascal and C. This data type is not predefined by FORTRAN, but you can simulate it by declaring an INTEGER*4 variable. This type specifies the number of planes in a bitmap; this value will fall between 0 and 31 inclusive.

gpr_$rhdm_pr_t

This is a predefined pointer type in both Pascal and C. Pascal predefines this type as

```
gpr_$rhdm_pr_t = ^PROCEDURE;
```

C predefines this type as

```
typdef void (*gpr_$rhdm_pr_t)();
```

In FORTRAN, you can simulate the gpr_$rhdm_pr_t data type by declaring an INTEGER*4 variable, and then using the IADDR function to store the starting address of a refresh operation. Regardless of the language, this type serves as a pointer to a routine that refreshes hidden display memory.

gpr_$rop_prim_set_elems_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies the primitives to which raster operations apply. In Pascal, FORTRAN, or C, gpr_$rop_prim_set_elems_t must be equal to one of the following predefined values:

gpr_$rop_blt
Apply raster operations to block transfers.

gpr_$rop_line
Apply raster operations to unfilled line primitives.

gpr_$rop_fill
Apply raster operations to filled primitives.

gpr_$rop_prim_set_t

This is a predefined set of gpr_$rop_prim_set_elems_t type in Pascal. C and FORTRAN do not support set types, but you can simulate this type by declaring a short int variable in C or an INTEGER*2 variable in FORTRAN. This set has 3 members. This type specifies the set of primitives that can have a raster operation established with gpr_$raster_op_prim_set. In addition, this set specifies the primitives for which a raster operation can be returned with gpr_$inq_raster_ops.

gpr_$rwin_pr_t

This is a predefined pointer type in both Pascal and C. Pascal predefines this type as

```
gpr_$rwin_pr_t=^PROCEDURE(IN unobscured:boolean;
                          IN pos_change:boolean)
```

C predefines this type as

```
typdef void (*gpr_$rwin_pr_t)();
```

In FORTRAN, you can simulate the gpr_$rwin_pr_t data type by declaring an INTEGER*4 variable, and then using the IADDR function to store the starting address of a refresh operation. Regardless of the language, this type serves as a pointer to a routine that refreshes a window.

gpr_$string_t

This is a predefined 256-element string in Pascal and C. It is not a predefined type in FORTRAN, but you can simulate this type by declaring a character*256 variable.

gpr_$trap_list_t

This is a predefined 10-element array of gpr_$trap_t type in Pascal and C. This data type is meant to serve as an example. You will probably want to define your own array of gpr_$trap_t with the appropriate number of elements for your application. FORTRAN does not predefine the gpr_$trap_list_t data type, but you can simulate it by declaring the following variable:

```
integer*2  gpr_$trap_list_t(10,4)
```

gpr_$trap_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 6-element array of INTEGER*2. This type specifies the coordinates of the top and bottom line segments of a trapezoid. The diagram below illustrates the gpr_$trap_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
top	gpr_$horiz_seg_t	1,2,3
bot	gpr_$horiz_seg_t	4,5,6

Description of Each Field:

top.x_coord_l
The left-hand x-coordinate of the top line.

top.x_coord_r
The right-hand x-coordinate of the top line.

top.y_coord
The y-coordinate of the top line.

bot.x_coord_l
The left-hand x-coordinate of the bottom line.

bot.x_coord_r
The right-hand x-coordinate of the bottom line.

bot.y_coord
The y-coordinate of the bottom line.

gpr_$triangle_fill_criteria_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 2-element array of INTEGER*2. This type specifies the filling criterion to use on polygons decomposed into triangles or polygons rendered with gpr_$render_exact. The diagram below illustrates the gpr_$triangle_fill_criteria_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
wind_type	gpr_$winding_set_t	1
winding_no	integer16/short int	2

Description of Each Field:

wind_type
The type of fill criterion to use in gpr_$winding_set_t format.

winding_no
The winding number to be used when the wind_type is gpr_$specific.

gpr_$triangle_list_t

This is a predefined 10-element array of gpr_$triangle_t type in Pascal and C. This data type is meant to serve as an example. You will probably want to define your own array of gpr_$triangle_t type with the appropriate number of elements for your application. FORTRAN does not predefine the gpr_$triangle_t data type, but you can simulate it by declaring the following variable:

```
integer*2  gpr_$triangle_t(10,7)
```

gpr_$triangle_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 7-element array of INTEGER*2. Specifies the coordinates of a triangle. The diagram below illustrates the gpr_$triangle_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
p1	gpr_$position_t	1,2
p2	gpr_$position_t	3,4
p3	gpr_$position_t	5,6
winding	integer16/short int	7

Description of Each Field:

p1.x_coord
The x-coordinate of point 1.

p1.y_coord
The y-coordinate of point 1.

p2.x_coord
The x-coordinate of point 2.

p2.y_coord
The y-coordinate of point 2.

p3.x_coord
The x-coordinate of point 3.

p3.y_coord
The y-coordinate of point 3.

winding
The winding number.

gpr_$version_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 2-element array of INTEGER*2. This type specifies the version number of an external bitmap header. The diagram below illustrates the gpr_$version_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
major	integer16/short int	1
minor	integer16/short int	2

Description of Each Field:

major
The major version number.

minor
The minor version number.

gpr_$winding_set_t

This is a predefined enumerated type in Pascal and C. FORTRAN does not support enumerated types, but you can simulate this type by declaring an INTEGER*2 variable. This type specifies a fill criterion. In Pascal, FORTRAN, or C, gpr_$winding_set_t must be equal to one of the following predefined values:

gpr_$parity
Apply a parity fill.

gpr_$nonzero
Apply a nonzero fill.

gpr_$specific
Fill areas with a specific winding number.

gpr_$window_list_t

This is a predefined 10-element array of gpr_$triangle_t type in Pascal and C. This data type is meant to serve as an example. You will probably want to define your own array of gpr_$window_t type with the appropriate number of elements for your application. FORTRAN does not predefine the gpr_$triangle_t data type, but you can simulate it by declaring the following variable:

```
integer*2  gpr_$window_list_t(10,4)
```

gpr_$window_t

This is a predefined record type in Pascal and a predefined structure type in C. FORTRAN does not support record/structure types, but you can simulate this type by declaring a 4-element array of INTEGER*2. This type defines a rectangular section of a bitmap. x_coord and y_coord specify the coordinates of the top left-hand corner of a rectangle. x_size and y_size specify the width and height of the rectangle. The diagram below illustrates the gpr_$window_t data type:

Name of Field	Data type of Field in Pascal/C	Element # in FTN array
window_base	gpr_$position_t	1,2
window_size	gpr_$offset_t	3,4

Description of Each Field:

window_base.x_coord
The x-coordinate of the top left-hand corner of the window.

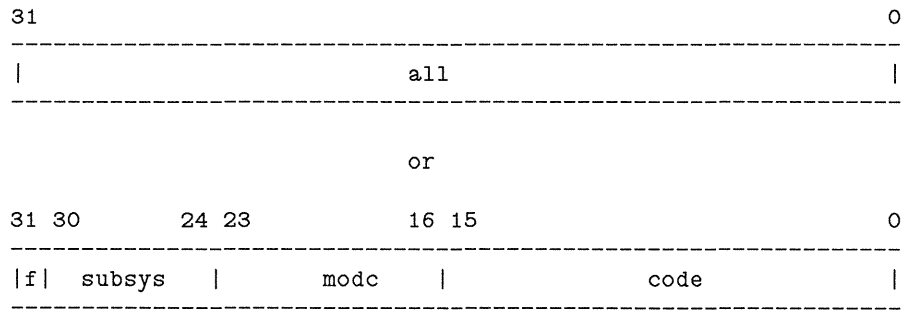
window_base.y_coord
The y-coordinate of the top left-hand corner of the window.

window_size.x_size
The width of the widow in pixels.

window_size.y_size
The height of the window in pixels.

status_$t

This is a predefined variant record in Pascal and a predefined union in C. FORTRAN does not predefine status_$t, but you can simulate it by declaring an INTEGER*4 variable. All GPR calls return a status code into a status_$t variable. You can read this status code to determine the success or failure of the preceding GPR call.



Description of Each Field:

all
All 32 bits in the status code.

fail
The fail bit. If this bit is set, the error was not within the scope of the module invoked, but occurred within a lower-level module (bit 31).

subsys
The subsystem that encountered the error (bits 24 - 30).

modc
The module that encountered the error (bits 16 - 23).

code
A signed number that identifies the type of error that occurred (bits 0 - 15).

Chapter 2
GPR Routines

This chapter lists all the user-callable GPR routines in alphabetical order. The description of each routine includes:

- An abstract of the routine's purpose.
- The format for calling the routine.
- A brief description of the purpose and data type of each parameter.
- A description of the routine's purpose.

If the description of a parameter contains the phrase "in XXX format", then XXX is a predefined data type in Pascal and in C (unless the data type is a set type). Since FORTRAN does not support predefined data types, we especially encourage FORTRAN programmers to refer back to Chapter 1 to learn how to simulate these predefined data types in FORTRAN. To aid FORTRAN programmers, many parameter descriptions contain a phrase that describes the data type in atomic terms, such as "This parameter is a 2-byte integer."

This manual does not contain any programming examples. However, we have printed many GPR programming examples in the *Programming With Domain Graphics Primitives* manual. In addition, many GPR programming examples are available on-line.

gpr_$acquire_display - Establishes exclusive access to the display hardware and the display driver.

FORMAT

```
unobscured := gpr_$acquire_display (status)
```

RETURN VALUE

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.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

While the display is acquired, the Display Manager cannot run. Hence, the program 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 (with gpr_$release_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. In the other display modes, the routine performs no operations.

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 - Transfers a single plane of any bitmap to all active planes of 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 containing 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 data type is 8 bytes long.

source_plane
: The identifier of the source plane to add, in gpr_$plane_t format. This is a 2-byte integer. Valid values are in the range 0 through hi_plane (where hi_plane is a parameter of gpr_$init).

dest_origin
: Start position (top left coordinate position) of the destination rectangle, in gpr_$position_t format. This data type is 4 bytes long. Coordinate values must be within the limits of the current bitmap, unless clipping is enabled.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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.

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 a set of default bitmap attribute settings, and returns the descriptor for the data structure.

FORMAT

```
gpr_$allocate_attribute_block (attrib_block_desc, status)
```

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.

USAGE

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. Possible values for width and height are 1 - 8192. This data type is four bytes long. See the GPR Data Types section for more information.

hi_plane_id
: Identifier of the highest plane which the bitmap will use, in gpr_$rgb_plane_t format. This is a 2-byte integer. Valid values are 0 - 31.

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. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

To establish an allocated bitmap as the current bitmap, use gpr_$set_bitmap.

To deallocate a bitmap, use gpr_$deallocate_bitmap. A program cannot use a bitmap after it is deallocated.

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,status)
```

INPUT PARAMETERS

size
: Bitmap width and height, in gpr_$offset_t format. This data type is 4 bytes long. The maximum size for a main-memory bitmap is 8192 x 8192.

hi_plane_id
: Identifier of the highest plane which the bitmap will use, in gpr_$rgb_plane_t format. This is a 2-byte integer. Valid values are 0 - 31.

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.

USAGE

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.

Use gpr_$set_bitmap to establish an allocated bitmap as the current bitmap.

Use gpr_$deallocate_bitmap to deallocate a bitmap. A program cannot use a bitmap after it is deallocated.

gpr_$allocate_buffer - Allocates a buffer bitmap in display memory having the same size and attributes as a specified display bitmap.

FORMAT

```
gpr_$allocate_buffer (primary_bitmap, buffer_bitmap, status)
```

INPUT PARAMETERS

primary_bitmap
: The descriptor of a bitmap in gpr_$bitmap_desc_t format. (The gpr_$init call will return this descriptor.) This is a 4-byte integer. You must specify a display bitmap; you cannot specify a hidden display memory bitmap, main memory bitmap, or external file bitmap.

OUTPUT PARAMETERS

buffer_bitmap
: Descriptor of the allocated bitmap, in gpr_$bitmap_desc_t format. This is a 4-byte integer.

status
: Completion status, in status_$t format.

USAGE

Use this call to establish a buffer bitmap for a double buffer application. The buffer bitmap will have the same rectangle properties as the primary bitmap. The buffer bitmap will also inherit the attribute block of the primary bitmap. (However, after creating the buffer bitmap, you may wish to allocate a separate attribute block for it.)

The buffer bitmap is invisible until you make it visible with the gpr_$select_display_buffer call.

Currently, the only node that permits double buffering is the DN590; furthermore, the DN590 permits only one buffer bitmap. If you try to allocate more than one buffer bitmap, GPR will return the following error code:

gpr_$no_more_fast_buffers

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 data type is 4 bytes long. The maximum size allowed for hidden display memory bitmaps is 224 bits by 224 bits.

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 memory, in gpr_$bitmap_desc_t format. This is a 4-byte integer.

status
: Completion status, in status_$t format.

USAGE

gpr_$allocate_hdm_bitmap allocates a bitmap in hidden display memory for programs running in a borrow or direct mode. You cannot allocate a hidden display memory bitmap if your program is running in frame mode.

In direct mode you must acquire the display before calling gpr_$allocate_hdm_bitmap. You acquire the display with the gpr_$acquire_display routine.

Use gpr_$deallocate_bitmap to deallocate a hidden display memory bitmap.

gpr_$arc_c2p - Draws an arc from the current position to the point where the arc intersects a user-defined ray.

FORMAT

```
gpr_$arc_2cp (center, p2, direction, option, status)
```

INPUT PARAMETERS

center
: The center of the arc, in gpr_$position_$t format. This data type is 4 bytes long.

p2
: A coordinate position used to define a ray that passes through the center of the arc, in gpr_$position_t format. The arc begins at the current position and ends where it intersects the ray. This data type is 4 bytes long.

direction
: The drawing direction for the arc in gpr_$arc_direction_t format. This data type is 2 bytes long. Possible values are gpr_$arc_ccw (counter-clockwise) and gpr_$arc_cw (clockwise).

option
: The choice of whether or not to draw an arc if the current position is coincident with the terminal point of the arc. This parameter is in gpr_$arc_option_t format. Possible values are gpr_$arc_draw_none and gpr_$arc_draw_full.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

The radius of the arc equals the distance from center to the current position.

After the arc is drawn, the point where the arc intersects the ray becomes the new current position.

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_$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 data type is 4 bytes long.

point_3
: The third point on the arc, in gpr_$position_t format. This data type is 4 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

It is geometrically possible to draw an arc through any three non-colinear points. The gpr_$arc_3p routine begins the arc at the current position, continues the arc through point_2, and completes the arc at point_3. After the arc is drawn, point_3 becomes the new current position.

You can use the gpr_$move command to re-establish the current position.

The call returns an error if any of the three points are equal or if the three points are colinear.

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)
```

RETURN 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.

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.

USAGE

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 data type is 8 bytes long.

source_plane
: Identifier of the single plane of the source bitmap to move, in gpr_$rgb_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. (The current limits are 31 for main memory bitmaps and 23 for display memory bitmaps.)

dest_origin
: Start position (top left coordinate position) of the destination rectangle, in gpr_$position_t format.

dest_plane
: Identifier of the plane of the destination bitmap, in gpr_$rgb_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.

USAGE

Both the source and destination bitmaps can be in any bitmap.

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.

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 data type is 4 bytes long.

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.

USAGE

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

gpr_$circle does not change the current position.

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 data type is 4 bytes long.

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.

USAGE

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

gpr_$circle_filled does not change the current position.

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.

FORMAT

```
gpr_$clear (color, status)
```

INPUT PARAMETERS

color

The color that all pixels in the current bitmap should be set to, 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 pixel format
0 - 16,777,215  for color displays in 24-bit pixel format
-2              for all displays.
```

OUTPUT PARAMETERS

status

Completion status, in status_$t format.

USAGE

The gpr_$clear routine sets every pixel in the current bitmap to the specified color. Don't be fooled by the name; gpr_$clear does not necessarily "clear" the bits to black. It actually sets all the bits to a particular color. You could, for example, use gpr_$clear to set all the bits in the current bitmap to purple.

If you specify the special number -2 for color, the system sets every bit in the current bitmap to the bitmap's background color. If the current bitmap is a main memory bitmap, hidden display memory bitmap, or external file bitmap, the background color is zero. If the current bitmap is a display memory bitmap in borrow mode, then the background color is zero (which is usually, but not always, black). If the current bitmap is a display memory bitmap in frame or direct mode, the background color is the same as that used for the window background color.

You can use gpr_$set_color_map to establish the correspondence between color map indexes and color values. This means that you can use gpr_$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.

gpr_$close_fill_pgon - Closes and fills the currently open polygon.

FORMAT

```
gpr_$close_fill_pgon (status)
```

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

gpr_$close_fill_pgon closes and fills the series of polygon boundaries created with the routines gpr_$start_pgon and gpr_$pgon_polyline.

Different decomposition techniques offer different rasterizations of polygons. For details, see the *Programming With DOMAIN Graphics Primitives* manual.

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 a gpr_$trap_list_t array of up to 10 elements.

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.

USAGE

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.

Note that gpr_$close_return_pgon does not work when the decomposition technique is gpr_$non_overlapping_tris.

gpr_$close_return_pgon_tri - Closes the currently open polygon and returns a list of triangles within its interior.

FORMAT

```
gpr_$close_return_pgon_tri (list_size, t_list, n_triangles, status)
```

INPUT PARAMETERS

list_size
: Maximum number of triangles that the routine is to return.

OUTPUT PARAMETERS

t_list
: Triangles returned. This is a gpr_$triangle_list_t array.

n_triangles
: Number of triangles that exist within the polygon interior. This is a 2-byte integer.

status
: Completion status, in status_$t format.

USAGE

gpr_$close_return_pgon_tri returns a list of triangles within a polygon interior that the graphics program can fill at a later time with the routine gpr_$multitriangle.

gpr_$close_return_pgon_tri returns a list of triangles when a polygon has been defined using gpr_$start_pgon and gpr_$pgon_polyline with the decomposition technique set to gpr_$non_overlapping_tris.

The n_triangles parameter is always the total number of triangles composing the polygon interior. If this number is greater than the list_size parameter, some triangles were left out of the t_list for lack of space.

Note that gpr_$close_return_pgon does not work when the decomposition technique is gpr_$non_overlapping_tris.

gpr_$color_zoom - Sets the magnification scale factor for a color display.

FORMAT

```
gpr_$color_zoom (xfactor, yfactor, status)
```

INPUT PARAMETERS

xfactor
: A 2-byte integer that denotes the magnification factor for the x-coordinate, in the range 1 through 16.

yfactor
: A 2-byte integer that denotes the magnification factor for the y-coordinate, in the range 1 through 16.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

The gpr_$color_zoom routine sets the magnification factor for all images drawn or BLTed into display memory on certain color nodes. By default, the magnification factor is 1 in the x direction and 1 in the y direction, meaning that the system will display graphics in the exact sizes you specify. For example, if you specify a 60 pixel by 60 pixel rectangle, then the system will display the rectangle as 60 pixels by 60 pixels. However, if you use gpr_$color_zoom to set the magnification factor to 3 in the x direction and 2 in the y direction, then the rectangle will be displayed as 180 pixels by 120 pixels. All future images will also be magnified unless you call gpr_$color_zoom a second time to set the magnification factors back to 1 and 1.

gpr_$color_zoom uses the integer zoom feature of the color hardware. gpr_$color_zoom always zooms from the upper-left corner of the display. Even if magnification causes all or part of the image to fall outside the bitmap dimensions, no error is returned.

The following restrictions apply to the gpr_$color_zoom routine:

- The call only works for programs running on a color node. If you call gpr_$color_zoom on a monochrome target, the system will return the error "Wrong display hardware".

- The call only works for programs running in borrow mode. If you call gpr_$color_zoom in a different display mode, the system will return the error "Must borrow display for this operation".

- If you specify a xfactor other than 1, then you cannot specify a yfactor of 1. However, if you specify a xfactor of 1, you can specify a yfactor other than 1.

- The DN570/570A and DN3000 nodes do not contain color zoom hardware. Therefore, if you specify values other than xfactor = 1 and yfactor = 1, then the system will return the error "Wrong display hardware?"

- The DN580/580T nodes permit limited use of the gpr_$color_zoom call. You cannot specify a xfactor or yfactor greater than 2.

The gpr_$inq_disp_characteristics routine returns the maximum magnification factors for the target node in the x and y dimensions.

Future Apollo nodes may or may not support color zoom hardware.

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)
```

RETURN VALUE

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_$set_obscured_opt and specified an option of either gpr_$ok_if_obs or gpr_$input_ok_if_obs.

OUTPUT PARAMETERS

event_type
: The type of event that occurred, in gpr_$event_t format. This is a 2-byte integer. One of the following values is returned:

```
gpr_$keystroke          Input from a keyboard
gpr_$buttons            Input from mouse or bitpad puck buttons
gpr_$locator            Input from a touchpad or mouse
gpr_$locator_update     Most recent 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. Its datatype is a character (char). This parameter is not modified for other events.

position
: The position on the screen or within the window at which graphics input occurred, in gpr_$position_t format. This data type is 4 bytes long.

status
: Completion status, in status_$t format.

USAGE

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.

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.

Unlike gpr_$event_wait, this call never releases the display.

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.

USAGE

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.

USAGE

To allocate a bitmap, use gpr_$init, gpr_$allocate_bitmap, gpr_$allocate_bitmap_nc, gpr_$open_bitmap_file, or gpr_$allocate_hdm_bitmap.

gpr_$deallocate_buffer - Deallocates a buffer bitmap.

FORMAT

```
gpr_$deallocate_buffer (primary_bitmap, buffer_bitmap, status)
```

INPUT PARAMETERS

primary_bitmap
: The descriptor of the primary bitmap in gpr_$bitmap_desc_t format. This is a 4-byte integer. (This descriptor was returned by gpr_$init.)

buffer_bitmap
: The descriptor of the buffer bitmap in gpr_$bitmap_desc_t format. This is a 4-byte integer. (This descriptor was returned by gpr_$allocate_buffer.)

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Use this call to deallocate (i.e., delete) the buffer bitmap you created with the gpr_$allocate_buffer call.

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. This is a 2-byte integer Specify only one of the following events:

gpr_$keystroke	Input from a keyboard
gpr_$buttons	Input from mouse or bitpad puck buttons
gpr_$locator	Input from a touchpad or mouse
gpr_$locator_update	Most recent 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

OUTPUT PARAMETERS

status
Completion status, in status_$t format.

USAGE

Following this call, no events of the given event type will be returned by gpr_$event_wait or gpr_$cond_event_wait.

In borrow 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 (x1, y1, x2, y2, status)
```

INPUT PARAMETERS

x1
: The x-coordinate of the top left-hand corner of the box. This is a 2-byte integer.

y1
: The y-coordinate of the top left-hand corner of the box. This is a 2-byte integer.

x2
: The x-coordinate of the bottom right-hand corner of the box. This is a 2-byte integer.

y2
: The y-coordinate of the bottom right-hand corner of the box. This is a 2-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the top left-hand and bottom right-hand 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 the program uses the pointer to access display memory.

FORMAT

```
gpr_$enable_direct_access (status)
```

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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_$locator            Input from a touchpad or mouse
gpr_$locator_update     Most recent 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
```

key_set
: The set of specifically enabled characters when the event class is enabled, in gpr_$keyset_t format. This parameter is specified for event types of gpr_$keystroke and gpr_$buttons. In Pascal, this is a set of characters. In FORTRAN and C this can be implemented as an eight element array of 4-byte integers.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

This routine specifies the type of event and event input for which gpr_$event_wait or gpr_$cond_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 separately 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.pas contains definitions for the non-ASCII keyboard keys in the range 128 - 255.

A group of calls is available for manipulating large sets in FORTRAN or C. The calls are: LIB_$INIT_SET, LIB_$ADD_TO_SET, LIB_$CLR_FROM_SET, and LIB_$MEMBER_OF_SET. These calls are fully described in *Programming with General System Calls*.

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.

For an exact cursor path use gpr_$locator with gpr_$set_cursor_position. Most applications can use gpr_$locator_update. With this value, GPR automatically tracks the most recent cursor location and gpr_$set_cursor_position is not needed.

gpr_$locator_update eliminates multiple locator events between gpr_$event_wait calls. Only one locator event will be delivered at a time, and the reported position will be the most recent one.

Regardless of the type(s) of events you enable, the only way to see the cursor is to call the gpr_$set_cursor_active routine.

gpr_$event_wait - Waits for an event.

FORMAT

```
unobscured := gpr_$event_wait (event_type, event_data, position, status)
```

RETURN VALUE

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_$set_obscured_opt and specified an option of either gpr_$ok_if_obs or gpr_$input_ok_if_obs.

OUTPUT PARAMETERS

event_type
: The type of event that occurred, in gpr_$event_t format. This is a 2-byte integer. One of the following predefined values is returned:

```
gpr_$keystroke          Input from a keyboard
gpr_$buttons            Input from mouse or bitpad puck buttons
gpr_$locator            Input from a touchpad or mouse
gpr_$locator_update     Most recent 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.

position
: The position on the screen or within the window at which graphics input occurred, in gpr_$position_t format. This data type is 4 bytes long.

status
: Completion status, in status_$t format.

USAGE

This routine suspends process execution until the occurrence of an event type enabled with 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.

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.

The display does not need to be acquired to call gpr_$event_wait.

If the display is acquired, gpr_$event_wait will implicitly release the display when the current process is waiting for an event to occur, or when an event that has not been enabled occurs and that event 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)
```

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.

USAGE

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.

USAGE

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 and allocates an initial bitmap.

FORMAT

```
gpr_$init (op_mode, unit, size, hi_plane_id, init_bitmap_desc, status)
```

INPUT PARAMETERS

op_mode
: The display mode for the program in gpr_$display_mode_t format. Possible values for this parameter are:

  gpr_$borrow
  : pseudo-color program borrows the full screen and the keyboard from the Display Manager and uses the display driver directly through GPR software. The initial bitmap will be stored in display memory.

  gpr_$borrow_rgb
  : true-color program borrows the full screen and the keyboard from the Display Manager and uses the display driver directly through GPR software. The initial bitmap will be stored in display memory.

  gpr_$borrow_nc
  : same as gpr_$borrow except that all the pixels are not set to zero. (The system does not clear the screen.) The initial bitmap will be stored in display memory.

  gpr_$borrow_rgb_nc
  : same as gpr_$borrow_rgb except that all the pixels are not set to zero. (The system does not clear the screen.) The initial bitmap will be stored in display memory.

  gpr_$direct
  : pseudo-color program borrows a window from the Display Manager instead of borrowing the whole display. The initial bitmap will be stored in display memory.

  gpr_$direct_rgb
  : true-color program borrows a window from the Display Manager instead of borrowing the whole display. The initial bitmap will be stored in display memory.

  gpr_$frame
  : pseudo-color program executes within a frame of a Display Manager Pad. Frame mode is not recommended because frame mode programs run more slowly than direct mode or borrow mode programs. The initial bitmap will be stored in display memory.

  gpr_$no_display
  : gpr allocates a bitmap in main memory. No graphics are displayed on the screen. The initial bitmap will be stored in main memory (not display memory) and the program can manipulate main memory bitmaps only.

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 (or the size of the frame, if op_mode equals gpr_$frame) in gpr_$offset_t format. The range of sizes you can specify depends on the op_mode.

If the op_mode is one of the four borrow modes, then you must set both dimensions of size to an integer between 1 and 8192 inclusive. If you provide bitmap dimensions smaller than the display memory of the node you are using, the size of the bitmap will match the dimensions you provide, If, however, you provide dimensions larger than the size of the display memory, the system will reduce the size of the initial bitmap to match the size of the display memory on your node. The origin of the bitmap is the top left corner of the screen.

If the op_mode is one of the two direct modes, then you must set both dimensions of size to integers between 1 and 8192 inclusive. If you provide dimensions smaller than the current display window, the system sets the size of the bitmap equal to the values you specified. If you provide dimensions larger than the display window, the system sets the size of the bitmap equal to the current size of the display window. However, if you grow the display window, then the bitmap will grow also, but cannot grow past the dimensions you specified. The origin of the bitmap is the top left corner of the display window.

If the op_mode is gpr_$frame, you must set both dimensions of size to integers between 1 and 32767 inclusive. For this mode, "size" specifies the size of both the frame and the initial bitmap. (In frame mode, the frame and the bitmap are the same size; see the *Programming With Domain Graphics Primitives* manual.

If the op_mode is gpr_$no_display, you must set both dimensions of size to integers between 1 and 8192 inclusive. The size that you specify will equal the size that the system allocates for a main memory bitmap.

hi_plane_id
Identifier of the bitmap's highest plane, in gpr_$rgb_plane_t format. This is a 2-byte integer. Valid values are:

```
For display memory bitmaps:

0      for monochromatic displays.
0 -  3 for 4-plane color displays.
0 -  7 for 8-plane color displays.
0 -  7 for the DN590 node in 8-plane hardware video mode.
0 - 23 for the DN590 node in 24-plane hardware video mode.


For main memory bitmaps:

0 - 31 for all displays
```

Programs running in gpr_$borrow_rgb or gpr_$direct_rgb mode should set hi_plane_id to 23.

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.

USAGE

The gpr_$init routine performs two separate actions. First, it initializes the graphics package, thus allowing you to make subsequent GPR calls. (The only GPR routines you can call before gpr_$init are gpr_$inq_config and gpr_$inq_disp_characteristics.) Second, GPR allocates a bitmap, usually in display memory.

Use the "RGB" modes for initializing true-color programs, and use the non-RGB modes for initializing pseudo-color programs. If you specify a RGB mode on a node other than the DN590, the system will return the status code

```
gpr_$wrong_display_hardware
```

If one program uses multiple windows, you must call gpr_$init for each window that uses GPR calls.

To use an imaging format, you must initialize the program in gpr_$borrow or gpr_$borrow_nc mode.

gpr_$inq_bitmap - Returns the descriptor of the current bitmap.

FORMAT

```
gpr_$inq_bitmap (bitmap_desc, status)
```

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.

USAGE

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 data type is 4 bytes long.

hi_plane_id
: The identifier of the bitmap's highest plane, in gpr_$rgb_plane_t format. This is a 2-byte integer. To find the number of planes in the bitmap, add one to hi_plane_id.

status
: Completion status, in status_$t format.

USAGE

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_file_color_map - Returns the specified entries from the external-bitmap color map.

FORMAT

```
gpr_$inq_bitmap_file_color_map (bitmap, start, entries, color, status)
```

INPUT PARAMETERS

bitmap
: The bitmap descriptor for the bitmap file in gpr_$bitmap_desc_t format. This is a 4-byte integer.

start
: The index of the first entry. This is a 2-byte integer.

entries
: The number of consecutive color-map entries to return. This is a 2-byte integer.

OUTPUT PARAMETERS

color
: The color values in UNIV gpr_$color_vector_t format. This is an array of long integers (4-byte integers).

status
: Completion status, in status_$t format.

USAGE

Each external bitmap is allocated its own color map. The external bitmap's color map is copied into the system color map whenever the external bitmap becomes the current bitmap.

You can inquire or change the values of the external bitmap's color map without making the external bitmap current.

Use gpr_$set_bitmap_file_color_map to change the values of an external bitmap's color map.

For the monochromatic display, the default start-index is 0. The number of entries is 2, and the color values are gpr_$black and gpr_$white. Dark has the value GPR_$BLACK, and bright has the value gpr_$white.

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.

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.

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.

USAGE

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. The parameter storage_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 parameter storage_ptr to access the screen bitmap on a monochrome system that uses a simulated color map, then pixels which are white have a pixel value of 1 and pixels that are black have a pixel value of 0, regardless of any calls to gpr_$set_color_map. In other words, the pixel value itself specifies the color of the pixel: the pixel value is not used as an index into the color map. On systems that have the color map in hardware, the pixel value is used as an index into the color map. The color of the pixel is determined by the color value in the color map.

On monochromatic devices, use gpr_$inq_disp_characteristics to determine whether the color map is simulated or in hardware. See the datatype gpr_$disp_char_t in Chapter 1 of this manual for more information.

If the cursor is active, the cursor pattern appears in the bitmap.

A program cannot use this routine on a bitmap which is a display manager pad (i.e., a frame mode bitmap).

gpr_$inq_bitmap_position - Returns the position of the upper left corner of the specified bitmap. This is normally the screen position; although, it does have some significance for main memory bitmaps.

FORMAT

```
gpr_$inq_bitmap_position(bitmap_desc, origin, status)
```

INPUT PARAMETERS

bitmap_desc
: The descriptor of the bitmap in gpr_$bitmap_desc_t format. This is a 4-byte integer.

OUTPUT PARAMETERS

origin
: The position of the upper left-hand corner of the bitmap in gpr_$position_t format. This data type is 4 bytes long.

status
: Completion status, in status_$t format.

USAGE

The bitmap position is different from the current position returned by the gpr_$inq_cp.

The gpr_$inq_bitmap_position routine is not meaningful if the bitmap is a display manager pad (i.e., a frame mode bitmap).

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 0 - 15.

status
: Completion status, in status_$t format.

USAGE

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.

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 character variable.

OUTPUT PARAMETERS

width
: The width parameter (in pixels) of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

status
: Completion status, in status_$t format.

USAGE

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; but 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 a 256-element array of 4-byte integers.

status
: Completion status, in status_$t format.

USAGE

To set the color map, use gpr_$set_color_map.

gpr_$inq_config - Returns the current display configuration.

FORMAT

```
gpr_$inq_config (config, status)
```

OUTPUT PARAMETERS

config

Display configuration, in gpr_$display_config_t format. This is a 2-byte integer. One of the following predefined values is returned:

Returned Value	Display Type
gpr_$bw_800x1024,	{ DN100, DN400 -- portrait }
gpr_$bw_1024x800,	{ DN3xx, DN4xx -- landscape }
gpr_$color_1024x1024x4,	{ DN600/660 2-board config }
gpr_$color_1024x1024x8,	{ DN600/660 3-board config }
gpr_$color_1024x800x4,	{ DN550/560 2-board config }
gpr_$color_1024x800x8,	{ DN550/560 3-board config }
gpr_$color_1280x1024x8,	{ DN580/590 }
gpr_$color1_1024x800x8,	{ DN570 }
gpr_$color2_1024x800x4,	{ DN3000C }
gpr_$bw_1280x1024,	{ DN3000M }
gpr_$color2_1024x800x8	{ DN3000E }

status

Completion status, in status_$t format.

USAGE

Use gpr_$inq_config to return the configuration constant of the node on which the program is executing. This constant tells you the size of the screen and the number of planes.

You can call gpr_$inq_config prior to calling gpr_$init.

gpr_$inq_constraints - Returns the clipping window and plane mask used for the current bitmap.

FORMAT

```
gpr_$inq_constraints (window, active, plane_mask, status)
```

OUTPUT PARAMETERS

window
: The clipping window, in gpr_$window_t format. This data type is 8 bytes long.

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.

USAGE

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, 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.

status
: Completion status, in status_$t format.

USAGE

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)
```

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.

USAGE

gpr_$inq_cp can be used to verify that the current position is at the desired location. If it is not, use gpr_$move to move the current position without drawing a line.

gpr_$inq_cursor - Returns information about the cursor.

FORMAT

```
gpr_$inq_cursor (curs_pat, curs_raster_op, active, position, origin,status)
```

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 of 2-byte integers. The default value is three. (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 data type is 4 bytes long.

origin
: The pixel currently set as the cursor origin, in gpr_$position_t format. This data type is 4 bytes long.

status
: Completion status, in status_$t format.

USAGE

Cursor position: If a program calls this routine when in borrow 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 or in direct mode, the x- and y-coordinates are relative to the top left corner of the frame or window.

Use one or more of the following routines to alter the cursor:

```
gpr_$set_cursor_pattern
gpr_$set_cursor_active
gpr_$set_cursor_position
gpr_$set_cursor_origin
```

Currently, a program cannot alter the cursor raster operation.

gpr_$inq_disp_characteristics - Allows the application program to obtain a variety of information about the nature of the actual display device or external bitmap if the program is operating in no-display mode.

FORMAT

```
gpr_$inq_disp_characteristics(op_mode, unit_or_pad, disp_len,
                              disp, disp_len_returned, status)
```

INPUT PARAMETERS

op_mode
: The op_mode (often called display mode) for the program in gpr_$display_mode_t format. (This is a 2-byte integer.) For example, if you specify gpr_$borrow, GPR will return information as if the program were running in pseudo-color borrow-display mode, regardless of the mode that you really initialized the program in.

unit_or_pad
: 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. For gpr_$no_display this parameter is ignored.

disp_len
: Size of the buffer (the disp parameter described below) in bytes provided by the calling program, which will contain the returned display or device information in bytes. For example, if the buffer is ten 16-bit words in length, the program gives 20 as the value of this parameter. No checking is (or can be) done to verify that this length is correct, so unpredictable results are obtained if the program gives a size that is larger than the actual size of the buffer. This parameter allows the calling program to request that less than the full set of characteristics be returned. It also allows the program to continue to function correctly if the list of returned characteristics is extended in the future.

OUTPUT PARAMETERS

disp
: Returned display device characteristics in gpr_$disp_char_t format. This data type is a record in Pascal, a structure in C, or an array in FORTRAN.

disp_len_returned
: Actual number of bytes of data returned in the disp parameter. This is a 2-byte integer. It will always be less than or equal to the disp_len input parameter value. Presently, the length of the full set of characteristics is 30 16-bit words, or 60 bytes. Therefore, 60 is the current maximum possible value for this parameter.

status
: Completion status, in status_$t format.

USAGE

Use gpr_$inq_disp_characteristics to determine your node's characteristics as it runs in a specified display mode. The characteristics include important information such as the size of the display screen and the number of planes. The call returns the characteristics into the disp parameter.

You can call gpr_$inq_disp_characteristics at any time in the program. In fact, it is good programming practice to call gpr_$inq_disp_characteristics prior to calling gpr_$init. By doing so, gpr_$inq_disp_characteristics will return values (such as bitmap size and hi_plane_id) that you can use when you call gpr_$init. In the future, we may extend the list of data items returned into disp as we release new display devices with new characteristics. However, programs written to use the existing set of characteristics will continue to operate correctly. Note that enumerated and set fields within disp will probably be extended in future releases; for example, we will probably add new controller types. Therefore, a program that depends on a particular controller type returned in disp may not work when run on a future node.

Note that calling gpr_$inq_disp_characteristics after gpr_$init has no effect on the current bitmap or its attributes. gpr_$inq_disp_characteristics is a purely descriptive call.

If you specify an op_mode of gpr_$direct or gpr_$direct_rgb, the call returns the values that are legal for a direct mode program at that instant. (Note that direct mode programs can potentially conflict with other direct mode programs when the display is not acquired by your program.)

Prior to this release, you could not call gpr_$inq_disp_characteristics when the screen was acquired. GPR now permits this. Note that the call returns information reflecting the status of the display when the call was made. Therefore, if you call gpr_$inq_disp_characteristics prior to acquiring the display, the returned information may not accurately reflect the future state of the window (since the window could have moved, grown, or been obscured).

gpr_$inq_draw_pattern - Returns the pattern used in drawing all line and curve primitives.

FORMAT

```
gpr_$inq_draw_pattern (repeat, pattern, length, status)
```

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 - 64.

status
: Completion status, in status_$t format.

USAGE

gpr_$inq_draw_pattern returns the current line pattern set explicitly with gpr_$set_draw_pattern.

This call will not return the line pattern set with gpr_$set_line_pattern. Use gpr_$inq_line_pattern to return the line pattern set with gpr_$set_line_pattern.

Use gpr_$set_draw_pattern or gpr_$set_line_pattern to specify a new line pattern. See gpr_$set_draw_pattern and gpr_$set_line_pattern for more information.

gpr_$inq_draw_value - Returns the color used for drawing lines.

FORMAT

```
gpr_$inq_draw_value (color, status)
```

OUTPUT PARAMETERS

color

The color 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	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.
-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 that the draw color is equal to the bitmap background color. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames and direct mode windows, this is the pixel value in use for the window background.

status

Completion status, in status_$t format.

USAGE

To set a new draw value, use gpr_$set_draw_value.

gpr_$inq_draw_width - Returns the line-width in pixels for all line and curve primitives.

FORMAT

```
gpr_$inq_draw_width (width, status)
```

OUTPUT PARAMETERS

width
: The current line width in pixels. This is a 2-byte integer.

status
: Completion status, in status_$t format.

USAGE

This routine returns the current line width used for lines and curves.

To set the line width use gpr_$set_draw_width.

gpr_$inq_fill_background_value - Returns the color of the background used for tile fills.

FORMAT

```
gpr_$inq_fill_background_value (color, status)
```

OUTPUT PARAMETERS

color

The color that the system is using 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	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a true-color program.
-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 that the fill background color is equal to the bitmap background color. 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.

USAGE

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)
```

OUTPUT PARAMETERS

pattern
: The descriptor of the bitmap containing the fill pattern, in gpr_$bitmap_desc_t format.

scale
: The number of times each bit in this pattern is to be replicated before proceeding to the next bit in the pattern in both the x and y directions. This is a 2-byte integer.

status
: Completion status, in status_$t format.

USAGE

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 32 x 32 pixels. 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 used to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
gpr_$inq_fill_value (color, status)
```

OUTPUT PARAMETERS

color

The current fill color, in gpr_$pixel_value_t format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays.
0 - 15	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.

status

Completion status, in status_$t format.

USAGE

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 horizontal spacing (in pixels) of the specified font. This is a 2-byte integer. Possible values are in the range -127 to 127.

status
: Completion status, in status_$t format.

USAGE

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)
```

OUTPUT PARAMETERS

format

Imaging format in gpr_$imaging_format_t configuration. This is a 2-byte integer. If you are using an interactive format, the returned value is gpr_$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.

USAGE

To set the imaging format, use gpr_$set_imaging_format.

gpr_$inq_line_pattern - Returns the pattern used in drawing lines.

FORMAT

```
gpr_$inq_line_pattern (repeat, pattern, length, status)
```

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 - 64.

status
: Completion status, in status_$t format.

USAGE

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 gpr_$dotted.

gpr_$inq_linestyle - Returns information about the current linestyle.

FORMAT

```
gpr_$inq_linestyle (style, scale, status)
```

OUTPUT PARAMETERS

style
: The style of line, in gpr_$linestyle_t format. This is a 2-byte integer. One of the following predefined values is returned:

  gpr_$solid for solid lines

  gpr_$dotted for dotted lines.

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.

USAGE

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_pgon_decomp_technique - Returns the mode which controls the algorithm used to decompose and rasterize polygons.

FORMAT

```
gpr_$inq_pgon_decomp_technique(decomp_technique,status)
```

OUTPUT PARAMETERS

decomp_technique
: Returns a mode which controls the algorithm used to decompose and render polygons, in gpr_$decomp_technique_t format. This is a 2-byte integer. Only one of the following predefined values is returned:

  gpr_$fast_traps
  : This is the default value on DN3XX/4XXs, DN550/560s, and DN6XXs which indicates that the faster but less precise algorithm is to be used. This is the only algorithm that existed prior to SR9.

  gpr_$precise_traps
  : This value indicates that a slower but more precise version of the decomposition algorithm is to be used.

  gpr_$non_overlapping_tris
  : This is the default value on DN570/580s and DN3000s which indicates that a triangle decomposition algorithm is to be used.

  gpr_$render_exact
  : This value indicates that the most precise rendering algorithm is to be used. It provides the best performance for rectilinear and axis aligned polygons, and it renders self-intersecting polygons more accurately than any of the other techniques in the following situation: when the intersection of two edges of the polygon is located at a noninteger.

status
: Completion status, in status_$t format.

USAGE

gpr_$inq_pgon_decomp_technique returns a mode setting, not an attribute.

gpr_$inq_raster_op_prim_set - Returns the primitive(s) which will be affected by the next gpr_$set_raster_op call, or the primitive(s) for which gpr_$inq_raster_op will return the current raster-op.

FORMAT

```
gpr_$inq_raster_op_prim_set (prim_set, status)
```

OUTPUT PARAMETERS

prim_set
: The set of primitives (lines, fills, and bit-block transfers) in gpr_$rop_prim_set_t format for which raster-ops can be set or inquired with gpr_$set_raster_op or gpr_$inq_raster_op, respectively.

status
: Completion status, in status_$t format.

USAGE

Use gpr_$inq_raster_op_prim_set to return the set of primitives that will be affected by gpr_$set_raster_op. Use gpr_$raster_op_prim_set to modify the set if necessary.

Use gpr_$inq_raster_op_prim_set to return the set of primitives that will have a raster op returned with gpr_$inq_raster_op

If prim_set contains the values gpr_$rop_line and gpr_$rop_fill, and the raster-ops for these operations are different, gpr_$inq_raster_op returns an error. When the values in prim_set have different raster-ops, call gpr_$raster_op_prim_set to establish the set with one value; then call gpr_$inq_raster_op.

gpr_$inq_raster_ops - Returns the raster operation for the primitives (lines, fills, and bit-block transfers) specified with gpr_$raster_op_prim_set.

FORMAT

```
gpr_$inq_raster_ops (raster_op, status)
```

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.

USAGE

To set a new raster operation for the primitives (lines, fills, and bit-block transfers) specified with gpr_$raster_op_prim_set, use gpr_$set_raster_op.

If the set of primitives established with gpr_$raster_op_prim_set have different raster-ops, this call returns an error.

If the set of primitives established with gpr_$raster_op_prim_set is empty, this call returns an error.

Use gpr_$inq_raster_op_prim_set to return the set of primitives established with gpr_$raster_op_prim_set.

When the values in the set of primitives established with gpr_$raster_op_prim_set have different raster-ops, call gpr_$raster_op_prim_set to establish the set with one value, then call gpr_$inq_raster_op.

gpr_$inq_refresh_entry - Returns two pointers: one to the procedure which refreshes the window; one to the procedure which refreshes hidden display memory.

FORMAT

```
gpr_$inq_refresh_entry (window_procedure, disp_mem_procedure, status)
```

OUTPUT 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. This is a pointer to a procedure.

status
: Completion status, in status_$t format.

USAGE

The returned routines apply to the current bitmap and current attribute block.

Applications can also direct the Display Manager to refresh the window automatically; see the routine gpr_$set_auto_refresh.

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 (in pixels) of the specified font. This is a 2-byte integer. Possible values are in the range -127 to 127.

status
: Completion status, in status_$t format.

USAGE

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 not included 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, direction, status)
```

OUTPUT PARAMETERS

font_id
: Identifier of the text font used for the current bitmap. This is a 2-byte integer.

direction
: The direction of movement from one text character position to the next in the current bitmap, in gpr_$direction_t format. This is a 2-byte integer. One of the following predefined values is returned:

```
gpr_$up,
gpr_$down,
gpr_$left,
gpr_$right
```

status
: Completion status, in status_$t format.

USAGE

To set a new text font for the current bitmap, use gpr_$set_text_font.

To change the direction of text, use gpr_$set_text_path.

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 a 256-element character array.

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 will occupy, in gpr_$offset_t format. This data type is 4 bytes long.

status
: Completion status, in status_$t format.

USAGE

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 width is the y-offset.

To change the direction of text, use gpr_$set_text_path.

gpr_$inq_text_offset - Returns the x- and y-offsets from the top left pixel of a string to the origin of the string's first character. This routine also returns the x- or y-offset to the pixel which is the new current position after the text is written with gpr_$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 a 256-element character array.

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 data type is 4 bytes long.

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.

USAGE

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.

gpr_$inq_text_path - Returns the direction for writing a line of text.

FORMAT

```
gpr_$inq_text_path (direction, status)
```

OUTPUT PARAMETERS

direction
: Direction for writing text, in gpr_$direction_t format. This is a 2-byte integer. One of the following predefined values is returned: gpr_$up, gpr_$down, gpr_$left, gpr_$right

status
: Completion status, in status_$t format.

USAGE

To set the current text path, use gpr_$set_text_path.

gpr_$inq_text_values - Returns the text color and the text background color used in the current bitmap.

FORMAT

```
gpr_$inq_text_values (text_color, text_bkgd_color, status)
```

OUTPUT PARAMETERS

text_color
: The color the system will use to write text, in gpr_$pixel_value_t format. This is a 4-byte integer.

text_bkgd_color
: The color the system will use as the background for text, in gpr_$pixel_value_t format. This is a 4-byte integer.

status
: Completion status, in status_$t format.

USAGE

To establish the text color, use gpr_$set_text_value. To establish the text background color, use gpr_$set_text_background_value.

gpr_$inq_triangle_fill_criteria - Returns the filling criteria used with polygons decomposed into triangles.

FORMAT

```
gpr_$inq_triangle_fill_criteria(fill_crit, status)
```

OUTPUT PARAMETERS

fill_crit
: Returns the filling criteria. This is a 2-byte integer. Possible values for this parameter are:

gpr_$parity	provides a means for filling polygons decomposed into triangles using an odd parity scheme. Regions filled in these polygons will match regions filled in polygons decomposed into trapezoids.
gpr_$nonzero	provides a means for filling all nonzero regions of a polygon.
gpr_$specific	provides a means for filling specific regions of a polygon. This is done by specifying a winding number. The only restriction is that regions with a winding number of zero cannot be filled.

status
: Completion status, in status_$t format.

USAGE

Use gpr_$pgon_decomp_technique to set a mode which controls the algorithm used to decompose polygons.

Use gpr_$set_triangle_fill_criteria to set the filling criteria used with polygons decomposed into triangles or for polygons rendered with the render exact algorithm.

For details on decomposition techniques, see the *Programming With DOMAIN Graphics Primitives* manual.

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
: 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 (see output parameters).

OUTPUT PARAMETERS

slots_total
: 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
: List of visible window sections. This is an array in gpr_$window_t format. This data type is eight bytes long. There is no set limit to the number of visible regions that may be returned.

status
: Completion status, in status_$t format.

USAGE

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_visible_buffer - Tells you whether it is the primary bitmap or the buffer bitmap that is currently being displayed.

FORMAT

```
gpr_$inq_visible_buffer (bitmap, status)
```

OUTPUT PARAMETERS

bitmap
: The descriptor, gpr_$bitmap_desc_t format, of either the primary bitmap or the buffer bitmap, whichever was last made visible. gpr_$bitmap_desc_t format is a 4-byte integer.

status
: Completion status, in status_$t format.

USAGE

Use the gpr_$inq_visible_buffer call to determine which bitmap is visible.

If you call gpr_$inq_visible_buffer before creating a buffer bitmap, the system will return the current bitmap (without returning an error).

If clipping is active, portions of both bitmaps may be visible at the same time. In this case, the value returned into the bitmap parameter will be the bitmap which was last made visible.

gpr_$inq_window_id - Returns the character that identifies the current bitmap's window.

FORMAT

```
gpr_$inq_window_id (character, status)
```

OUTPUT PARAMETERS

character
: The character that identifies the current bitmap's window.

status
: Completion status, in status_$t format.

USAGE

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 end point supplied. The current position is updated to the end point.

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.

USAGE

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. If you supply a relative pathname, the system will search for the font in the current directory and then in the /sys/dm/fonts directory.

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. Available fonts are listed in the directory /sys/dm/fonts.

status
: Completion status, in status_$t format.

USAGE

Use the font-id returned from this file as input for gpr_$set_text_font.

You can call gpr_$load_font_file multiple times without unloading fonts. However, if you do want to unload a font, call the gpr_$unload_font_file routine.

gpr_$move - Sets the current position to the given 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.

USAGE

The current position is the starting point for many drawing and text operations.

gpr_$move does not draw any lines.

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 coordinate positions in gpr_$coordinate_array_t format. This is an array of 2-byte integers. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive coordinate positions in gpr_$coordinate_array_t format. This is an array of 2-byte integers. 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.

USAGE

gpr_$multiline alternately moves to new positions 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.

If you specify an odd number of coordinate positions, then the system will use the last point as the new current position (but will not use it to draw a line).

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
: Trapezoids to fill, in gpr_$trap_list_t format. This data type is 12 bytes long.

trapezoid_number
: Number of trapezoids to fill. This is a 2-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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.

Different decomposition techniques offer different rasterizations of polygons. For details, see the *Programming With DOMAIN Graphics Primitives* manual.

gpr_$multitriangle - Draws and fills a list of triangles in the current bitmap.

FORMAT

```
gpr_$multitriangle (t_list, n_triangles, status)
```

INPUT PARAMETERS

t_list
: Triangles to fill in gpr_$triangle_list_t format. This data type is a variable size array where each element of the array contains 14 bytes.

n_triangles
: Number of triangles to fill. This is a 2-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

This call fills a list of triangles with the color/intensity value specified with gpr_$set_fill_value.

To retrieve the current fill value, use gpr_$inq_fill_value.

When entering coordinates for each triangle, you must set a winding number. The winding number must agree with filling criterion established with gpr_$set_triangle_fill_criteria. For example, if the filling criterion is gpr_$parity, the winding number of triangles to be filled must be odd. The default filling criterion is gpr_$parity.

Individual triangles can be assigned different winding numbers making it possible to fill specific triangles in the list using gpr_$set_triangle_fill_criteria.

Different decomposition techniques offer different rasterizations of polygons. For details, see the *Programming With DOMAIN Graphics Primitives* manual.

gpr_$open_bitmap_file - Opens (for creating or accessing) a bitmap stored on disk.

FORMAT

```
gpr_$open_bitmap_file (access, filename, filename_size, version, size, groups,
                       group_header, attribs, bitmap, created, status)
```

INPUT PARAMETERS

access
: One of four ways to access external bitmap objects, in gpr_$access_mode_t format. This is a 2-byte integer. Specify one of the following values:

  gpr_$create — Allocates a new file on disk for storage of a graphic image.

  gpr_$update — Allows you to modify a previously created file or create a new one.

  gpr_$write — allows you to write to an existing file.

  gpr_$readonly — allows you to read a previously created file.

filename
: The pathname of the bitmap file, in NAME_$PNAME_T format.

filename_size
: The length of the filename. This is a 2-byte integer. If you specify 0 when creating the file, the system will create a temporary file.

INPUT/OUTPUT PARAMETERS

version
: The version number on the header of the external bitmap file, in gpr_$version_t format. This is a two-element array of two 2-byte integers: a major version number and a minor version number. Currently, version is not used and is always returned as major version 1, minor version 1. If you specify an access other than gpr_$create, GPR ignores the value specified for version and returns the values allocated when the file was created.

size
: Bitmap width and height, in gpr_$offset_t format. This is a two-element array of 2-byte integers. The first element is bitmap width, in raster units; the second element is the bitmap height, in raster units. Possible values for x are 1-4096; possible values for y are 1-4096. You can get the bitmap width and height by calling gpr_$inq_disp_characteristics. If you are not creating the file, GPR ignores the value specified for size and returns the values allocated when the file was created.

groups
The number of groups in external bitmaps. This is a 2-byte integer. Possible values are 1..(gpr_$max_bmf_group +1). Currently, a bitmap can contain only 1 group. If you are not creating the file, GPR ignores the value specified for groups and returns the value allocated when the file was created.

group_header
Description of the external bitmap, in gpr_$bmf_group_header_array_t format. This is an array [0..gpr_$max_bmf_group] of gpr_$bmf_group_header_t. A description of the fields in a group header and the possible values are listed below. If you are not creating the file, GPR ignores all of the values specified in group_header, and returns the values allocated when the file was created.

N_SECTS
This is a 2-byte integer representing the number of sections in the group. Previously, this value had to be set equal to the number of planes on the target node. Now, permissable values range between 1 and 8 inclusive.

PIXEL_SIZE
This is a 2-byte integer representing the number of bits per pixel in each section of a group. Previously, this value had to be 1. Now, you can set it to any value from 1 to 32.

ALLOCATED_SIZE
This is a 2-byte integer representing the number of bits that the system uses to store the value of one pixel. The only legal values for ALLOCATED_SIZE are 0, 1, 8, 16, and 32. Choosing 0 means that the system will calculate ALLOCATED_SIZE for you. If you choose a number other then 0, than the value you choose must be greater than or equal to PIXEL_SIZE.

BYTES_PER_LINE
This is a 2-byte integer representing the number of bytes in one row of one plane of the bitmap. The value must be a multiple of 4 large enough to contain all the bytes in one line. If you set BYTES_PER_LINE to 0 when creating the file, GPR will perform the necessary calculations and return the appropriate value into the parameter.

BYTES_PER_SECT
The number of BYTES_PER_LINE multiplied by the height of the bitmap. This value must then 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. BYTES_PER_SECT is a 4-byte integer. If you set BYTES_PER_SECT to 0 when you create the file, then GPR will perform the necessary calculations and return the correct value into the parameter. BYTES_PER_SECT is not necessarily a multiple of BYTES_PER_LINE. 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.

STORAGE_OFFSET
GPR returns this UNIV_PTR parameter which points to the beginning of the group storage area.

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.

USAGE

This release contains several important improvements to the creation and access of external bitmaps. Note that existing external bitmap programs will not break as a result of these improvements.

In previous releases, a section was equivalent to one plane of an external bitmap. Therefore, if your node contained eight planes, you would store the external bitmap in eight different sections. In practical terms, you had to set N_SECTS to the number of planes and PIXEL_SIZE to 1. Furthermore, you had to use the gpr_$write_pixels and gpr_$read_pixels calls to write to and read from the external bitmap. These calls are not as fast as the blt calls.

This release introduces pixel-oriented bitmaps. You can now store an entire external bitmap in one section, even if the node contains many planes. In other words, a pixel-oriented bitmap stores the value of one pixel in consecutive bits, instead of scattered around a disk file. Furthermore, you can use the blt calls to move data between the external bitmap and any other kind of bitmap (e.g., the display bitmap). You can create clipping windows in an external bitmap. The net result of these changes is that the GPR system can display an external bitmap faster than in previous releases.

Drawing, fill, and text operations cannot be performed directly to a pixel-oriented bitmap. You can, however, do these operations to the display bitmap and then blt the display bitmap to a pixel-oriented bitmap.

For example, suppose you want to create an external bitmap for an 8-plane node. In this case, we suggest setting N_SECTS to 1 and PIXEL_SIZE to 8.

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_$create as the access parameter and you specify a filename that already exists, the file is superseded only if it is a bitmap file. If the file is not a bitmap file, you get the error message "name_$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.

gpr_$pgon_decomp_technique - Sets a mode which controls the algorithm used to decompose and render polygons.

FORMAT

```
gpr_$pgon_decomp_technique(decomp_technique,status)
```

INPUT PARAMETERS

decomp_technique

Sets a mode that controls the algorithm used to decompose and render polygons in gpr_$decomp_technique_t format. This is a 2-byte integer. Specify only one of the following predefined values:

gpr_$fast_traps

This is the default value on DN3XX, DN4XX, DN550/560, DN600/660 which indicates that the fast, but less precise, algorithm is to be used. This is the only algorithm that existed prior to SR9.

gpr_$precise_traps

This value indicates that a slower, but more precise, version of the trapezoid decomposition algorithm is to be used.

gpr_$non_overlapping_tris

This is the default value on the following models: DN570/570A/580 and DN3000.

gpr_$render_exact

This value indicates that the most precise rendering algorithm is to be used. It provides the best performance for rectilinear and axis-aligned polygons, and it renders self-intersecting polygons more accurately than any of the other techniques in the following situation: when the intersection of two edges of the polygon is located at a noninteger.

OUTPUT PARAMETERS

status

Completion status, in status_$t format.

USAGE

gpr_$pgon_decomp_technique establishes a mode setting, not an attribute. Setting the decomposition technique applies to all polygons drawn during a particular session of GPR (within a gpr_$init and gpr_$terminate), not just the polygons drawn in the current bitmap.

Polygons without self-crossing and "normal" self-crossing polygons work with the gpr_$fast_traps setting. Polygons with multiple self-crossings and/or vertices in close proximity may not be filled correctly with the gpr_$fast_traps setting. Fill these

polygons using the gpr_$precise_traps, gpr_$non_overlapping_tris, or gpr_$render_exact setting.

See *Programming with DOMAIN Graphics Primitives* for information on decomposition and rendering.

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. The gpr_$coordinate_array_t type, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. The gpr_$coordinate_array_t type, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. 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.

USAGE

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 , by (2) gpr_$close_return_pgon and gpr_$multitrapezoid, or by (3) gpr_$close_return_pgon_tri and gpr_$multitriangle. The lines are not drawn on the screen until the polygon is filled in by either routines (1), (2), or (3) above. To draw an unfilled polygon, 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 data type is 8 bytes long.

dest_origin
: Start position (top left coordinate position) of the destination rectangle, in gpr_$position_t format. This data type is 4 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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. If you specify a 32-plane main memory bitmap as the source bitmap and a 24-plane display memory bitmap as the destination, then the system will BLT only the first 24 planes of the main memory bitmap.

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.

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: drawing begins at the current position, draws to the first given coordinate position, then sets the current position to the first given position. This is repeated for all given positions.

FORMAT

```
gpr_$polyline (x, y, npositions, status)
```

INPUT PARAMETERS

x
: List of the x-coordinates of all the successive positions. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. 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.

USAGE

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.

gpr_$raster_op_prim_set - Specifies the primitive(s) which will be affected by the next gpr_$set_raster_op call, or the primitive(s) for which gpr_$inq_raster_op will return the current raster-op.

FORMAT

```
gpr_$raster_op_prim_set (prim_set, status)
```

INPUT PARAMETERS

prim_set
: The set of primitives (lines, fills, and bit-block transfers) in gpr_$rop_prim_set_elems_t format for which raster-ops can be set or inquired with gpr_$set_raster_op or gpr_$inq_raster_op, respectively.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Use gpr_$raster_op_prim_set to specify which primitives will be affected when a raster operation is set with gpr_$set_raster_op. For example, if prim_set contains the values gpr_$rop_line and gpr_$rop_fill, only line and fill raster operations will be affected with the next call to gpr_$set_raster_op.

Use gpr_$raster_op_prim_set to specify the primitives for which gpr_$inq_raster_op will return the raster-op. If the members of the set have different raster-ops or if the set is empty, an error message is returned.

Raster-ops for lines, fills, and blts can be different at the same time by making successive calls to gpr_$raster_op_prim_set and gpr_$set_raster_op.

The default prim_set contains gpr_$rop_line and gpr_$rop_blt.

gpr_$rop_line affects the following routines: gpr_$line, gpr_$polyline, gpr_$multiline, gpr_$draw_box, gpr_$circle, and gpr_$arc_3p.

gpr_$rop_fill affects the following routines: gpr_$triangle, gpr_$multitriangle, gpr_$trapezoid, gpr_$close_fill_pgon, gpr_$circle_filled, and gpr_$rectangle.

gpr_$rop_blt affects the following routines: gpr_$bit_blt, gpr_$pixel_blt, and gpr_$additive_blt.

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 data type is 8 bytes long.

OUTPUT PARAMETERS

pixel_array
: An array from which to read pixel values in gpr_$pixel_array_t format. This is an array of 4-byte integers. You can specify an array of up to 131,073 elements.

status
: Completion status, in status_$t format.

USAGE

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.

A program cannot use this routine on a bitmap corresponding to a Display Manager frame.

A program cannot read pixel values in imaging formats.

If you read more pixels than there are in pixel_array, unpredictable results may occur.

gpr_$rectangle - Draws and 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 data type is 8 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

gpr_$rectangle fills in a rectangle with the color specified with gpr_$set_fill_value. To retrieve the current fill value, use gpr_$inq_fill_value.

To draw an unfilled rectangle use gpr_$draw_box or gpr_$polyline.

gpr_$release_display - Decrements a counter associated with the number of times a display has been acquired.

FORMAT

```
gpr_$release_display (status)
```

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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.

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_$remap_color_memory - Defines the plane in color display memory for which a pointer will be returned when using gpr_$inq_bitmap_pointer. This allows a single plane of color display memory to be accessed directly.

FORMAT

```
gpr_$remap_color_memory (plane, status)
```

INPUT PARAMETERS

plane
: The plane in color display memory in gpr_$rgb_plane_t. This is a 2-byte integer. A pointer can be returned to the plane using gpr_$inq_bitmap_pointer. Valid values are 0 - 23.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

When accessing color display memory directly (i.e. by dereferencing the pointer returned by gpr_$inq_bitmap_pointer), the program can access only one plane at a time. This is unlike access to multi-plane memory bitmaps, in which the first scan line of a plane immediately follows the last scan line of the previous plane in virtual memory, or access to bitmaps stored in bitmap files where bytes_per_section specifies the address difference between planes. Therefore, a program must use gpr_$remap_color_memory to establish which plane of color display memory will be accessible through the "storage_ptr" returned by gpr_$inq_bitmap_pointer.

gpr_$remap_color_memory_1 - Defines the plane in hidden color display memory for which a pointer is returned when using gpr_$inq_bitmap_pointer. This allows direct access to a single plane of color display memory.

FORMAT

```
gpr_$remap_color_memory_1 (plane, status)
```

INPUT PARAMETERS

plane
: The plane in hidden color display memory in gpr_$rgb_plane_t. This is a 2-byte integer. A pointer can be returned to the plane using gpr_$inq_bitmap_pointer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

gpr_$remap_color_memory_1 allows access to the normally hidden frame 1 of color display memory. gpr_$remap_color_memory allows access to frame 0.

gpr_$remap_color_memory_1 returns an error on the following machine models: DN570/570A/570-T/580/580-T/3000.

gpr_$replicate_font - Creates and loads a modifiable copy of a font.

FORMAT

```
gpr_$replicate_font (font_id, replicated_font_id, status)
```

INPUT PARAMETERS

font_id
: Identifier of the original text font. This is a 2-byte integer.

OUTPUT PARAMETERS

replicated_font_id
: Identifier of the copied text font. This is a 2-byte integer.

status
: Completion status, in status_$t format.

USAGE

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 color display memory is visible.

FORMAT

```
gpr_$select_color_frame (frame, status)
```

INPUT PARAMETERS

frame
: This is a 2-byte integer. Denotes which frame is to be visible. Possible values are zero or one. Normally, frame 0 is visible.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

gpr_$select_color_frame returns an error if any value other than 0 is entered on the following models: DN570/570A/580 and DN3000.

gpr_$select_display_buffer - Switches the buffers in a double buffering program, so that the displayed buffer becomes invisible and the invisible buffer becomes displayed.

FORMAT

```
gpr_$select_display_buffer(display_desc,option_desc,option_val,option,status)
```

INPUT PARAMETERS

display_desc
: The descriptor of the bitmap you want displayed in gpr_$bitmap_desc_t format. This is a 4-byte integer. You must specify the descriptor of either the primary bitmap or the buffer bitmap.

option_desc
: The descriptor of the "other" bitmap in gpr_$bitmap_desc_t format. This is a 4-byte integer. You must specify the descriptor of either the primary bitmap or the buffer bitmap, whichever one you did not specify in display_desc. That is, if display_desc contains the descriptor of the primary bitmap, then you must set option_desc equal to the descriptor of the buffer bitmap, and vice-versa. If you set the option parameter equal to gpr_$undisturbed_buffer, then the system ignores the value of option_desc.

option_val
: The color value that the bitmap specified in option_desc should be cleared to. The color value is a 4-byte integer. This parameter only has meaning if you set option (the next parameter) to gpr_$clear_buffer. If option is set to some other value, then the system ignores option_val.

option
: The action to take on the bitmap specified by option_desc in gpr_$double_buffer_option_t format. This is a 2-byte integer. Possible values for this parameter are:

  gpr_$clear_buffer
  : gpr sets every pixel in the bitmap to the color specified by option_val.

  gpr_$undisturbed_buffer
  : gpr does not change the value of any pixel in the specified bitmap.

  gpr_$copy_buffer
  : gpr copies the value of every pixel in the display_desc bitmap to every pixel in the option_desc bitmap. Use this option for incremental rendering.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

The gpr_$select_display_buffer call performs two separate actions. First, it lets you choose the bitmap (primary or buffer) to display. Second, it lets you take action on the bitmap that doesn't get displayed. Use display_desc to choose the bitmap to display. Use option_desc, option_val, and option to take action on the bitmap that doesn't get displayed.

Use gpr_$copy_buffer when you want to build a figure based on the previous figure. It is a very useful option for animation.

You must create a buffer bitmap with the gpr_$allocate_buffer call before calling gpr_$select_display_buffer.

This call only switches the pixels inside the clip window. The other pixels are unaffected by the gpr_$select_display_buffer call. Thus with clipping active, the screen can simultaneously display portions from both bitmaps.

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
: 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.

USAGE

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.

This routine may release and reacquire the display if the events enabled in the current and new attribute blocks are different.

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.

USAGE

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.

gpr_$auto_refresh 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
: A unique bitmap descriptor, in gpr_$bitmap_desc_t format. This is a 4-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

The program can obtain the bitmap descriptor by using gpr_$inq_bitmap.

After a bitmap is established using gpr_$set_bitmap or gpr_$init, 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 data type is 4 bytes long.

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.

USAGE

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_id
Monochromatic displays			
1024 x 800 screens	1024	1024	0
800 x 1024 screens	1024	1024	0
1280 x 1024 screens	2048	1024	0
Color display--Interactive format			
4-plane DN550/560/600/660	1024	2048	3
8-plane DN550/560/600/660	1024	2048	7
4-plane DN3000	1024	1024	3
8-plane DN3000	1024	1024	7
DN570	1024	1024	7
DN580	1504	1024	7
DN590	1504	1024	23

The system uses certain areas of hidden display memory to store fill constants, fonts, and cursor patterns. Suppose you use gpr_$set_bitmap_dimensions to expand the bitmap to include parts of hidden display memory. If you then write over parts of hidden display memory, you run the risk of overwriting the fill constants, fonts, or cursor patterns.

gpr_$set_bitmap_file_color_map - Establishes new values for the external-bitmap color map.

FORMAT

```
gpr_$set_bitmap_file_color_map (bitmap, start, entries, color, status)
```

INPUT PARAMETERS

bitmap
: The bitmap descriptor for the bitmap file in gpr_$bitmap_desc_t format. This is a 4-byte integer.

start
: The index of the first entry to be modified. This is a 2-byte integer.

entries
: The number of consecutive entries to be modified. This is a 2-byte integer.

color
: The color values in UNIV gpr_$color_vector_t format. This is an array of 32-bit integers.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Each external bitmap is allocated its own color map. The external bitmap's color map is copied into the system color map whenever the external bitmap becomes the current bitmap.

You can inquire or change the values of the external bitmap's color map without making the external bitmap current.

For the monochromatic display, the default start-index is 0. The value of entries is 2, and the color 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.

Use gpr_$inq_bitmap_file_color_map to return the values of an external-bitmap's color map.

gpr_$set_character_width - Specifies the width of the specified character in the specified modifiable 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 character variable.

width
: The width of the specified character in pixels. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

To retrieve a character's width, use gpr_$inq_character_width.

The initial character widths are defined in the font file.

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_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 data type is 8 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

The default clip window is the entire bitmap. Use the gpr_$set_clip_window to set a nondefault clip window. Note that the clip window does not become activated until you call gpr_$set_clipping_active.

In direct mode, the clip window and coordinate origin are relative to the the upper left-hand corner of the window.

A clip window cannot be made larger than the dimensions specified for a bitmap. For applications that run in windows that are dynamically enlarged, specify the size parameter of gpr_$init to be the size of the display. In this way, the clip rectangle will automatically be enlarged with the window whenever the window is enlarged.

Pixels outside the clip window in the current bitmap are not modified by subsequent operations.

To request the dimensions of the current clip window, use gpr_$inq_constraints.

This call is not allowed on the bitmap corresponding to the Display Manager frame.

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.

USAGE

To specify a clipping window, use the routine gpr_$set_clip_window.

Initially, in borrow mode, the clip window is disabled. In direct mode, the clip window is enabled and clipped to the size of the window. Clipping cannot be enabled in a bitmap corresponding to a Display Manager frame.

To inquire whether the clip window is enabled, use gpr_$inq_constraints.

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-plane format
1 - 256	For color displays in 8-plane format

values
: The color table, stored as an array of 4-byte integers. The gpr_$color_vector_t type is an example of such an array, though your array need not be this large.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Use the gpr_$set_color_map routine to change one, some, or all of the available slots in the color chart. See the *Programming With DOMAIN Graphics Primitives* manual for details on color maps.

In general, you do not have to call gpr_$set_color_map when running in an RGB mode because the system automatically loads a linear ramp color map for you. If you specify the Display Manager command CDM -p 1 and initialize the display mode with gpr_$direct, then calling gpr_$set_color_map will have no effect. In gpr_$borrow_rgb mode, you can call gpr_$set_color_map to alter the linear ramp color map (perhaps to perform a gamma correction).

On monochromatic displays, the color map is either simulated or in hardware. If the color map is simulated, then the pixel value 1 always corresponds to white and the pixel value 0 always corresponds to black. Calling gpr_$set_color_map on such a machine will not change these correspondences. However, if the color map is in hardware, then the system does use the pixel value as an index into the color map. In this instance, you can use gpr_$set_color_map to change 1 to black and 0 to white or vice-versa. By default, the start_index is 0, n_entries is 2, and the values are gpr_$black in the first array element

and gpr_$white in the second array element. Dark has the value gpr_$black, and bright has the value gpr_$white. If the program provides fewer than two values, or if the first two values are the same (both black or both white), the routine returns an error. To determine whether the color map is simulated or in hardware, call the gpr_$inq_disp_characteristics routine. This routine will return the answer into the invert field of the "disp" record/structure in Pascal or C (or element 29 of the "disp" array in FORTRAN).

In gpr_$direct mode, you must acquire the display before calling gpr_$set_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 for move, draw, text, fill, 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 data type is 4 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

To retrieve the current coordinate origin, use gpr_$inq_coordinate_origin.

The default coordinate origin is (0,0).

In direct mode, the clip window and coordinate origin are relative to the the upper left-hand corner of the window.

You cannot use gpr_$set_coordinate_origin 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
: Boolean (logical) value that specifies whether to display the cursor. Set the parameter to true if you want 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.

USAGE

Initially, the cursor is not displayed.

To inquire whether the cursor is currently displayed, use gpr_$inq_cursor.

A program may call this routine only while operating in borrow mode 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 data type is 4 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

A program uses gpr_$set_cursor_origin to designate one pixel in the cursor pattern 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.

USAGE

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 data type is 4 bytes long. 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. When running in frame mode, the x- and y-coordinates must fall within the range 0 to 32767 inclusive. In direct mode, the x- and y-coordinates must fall within the size of the window. In borrow mode, the x- and y-coordinates must fall within the size of display memory (as documented in the *Programming With DOMAIN Graphics Primitives* manual).

OUTPUT PARAMETERS

status

Completion status, in status_$t format. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

Cursor position: If a program calls this routine when in borrow 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.

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_pattern - Specifies the line pattern to use in drawing all line and curve primitives.

FORMAT

```
gpr_$set_draw_pattern (repeat_count, pattern, length, status)
```

INPUT PARAMETERS

repeat_count
: The replication factor for each bit in the pattern. This is a 2-byte integer. Specifying a value of 0 results in a solid line.

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. Specifying a value of 0 results in a solid line.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

All line and curve primitives use the pattern/style most recently defined by gpr_$set_draw_pattern. 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 set a line pattern with gpr_$set_linestyle or gpr_$set_line_pattern; however, once you have used gpr_$set_draw_pattern to set a line pattern, you must continue to use gpr_$set_draw_pattern to set line patterns. Any subsequent calls to gpr_$set_linestyle or gpr_$set_line_pattern results in an error.

gpr_$set_linestyle and gpr_$set_line_pattern sets a line pattern only for lines and splines: the line pattern does not affect other curved primitives.

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 bit of entry 4.

Use gpr_$inq_draw_pattern to retrieve the current draw pattern. This routine returns the pattern set explicitly with gpr_$set_draw_pattern.

gpr_$inq_line_pattern returns an error if the draw pattern was set with gpr_$draw_pattern.

You cannot set the line pattern for arcs if your application sets a line pattern with gpr_$set_line_pattern.

gpr_$set_draw_value - Specifies the color/intensity value to use to draw lines.

FORMAT

```
gpr_$set_draw_value (color, status)
```

INPUT PARAMETERS

color

The color to be 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	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.
-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 sets the drawing value equal to the color of the bitmap 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.

USAGE

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.

gpr_$set_draw_width - Sets the line width in pixels for line and curve primitives.

FORMAT

```
gpr_$set_draw_width (width, status)
```

INPUT PARAMETERS

width
: The line width in pixels for all line/curve primitives. This is a 2-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Use the draw-width attribute to establish a line width in pixels. If the line width is even, the extra pixels are added on the top or left-hand side of the line.

Use gpr_$inq_draw_width to retrieve the current line width.

gpr_$set_fill_background_value - Specifies the color/intensity value used for drawing the background of tile fills.

FORMAT

```
gpr_$set_fill_background_value (color, status)
```

INPUT PARAMETERS

color

The color to be 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	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.
-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 sets the tile file color equal to the color of the bitmap 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. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

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.

USAGE

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 to use to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
gpr_$set_fill_value (index, status)
```

INPUT PARAMETERS

color

The color to be used in fill operations, in gpr_$pixel_value_t format. This is a 4-byte integer. The default fill value is 1. Valid values are:

0 - 1	For monochromatic displays.
0 - 15	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.

OUTPUT PARAMETERS

status

Completion status, in status_$t format.

USAGE

To retrieve the current fill value, use gpr_$inq_fill_value.

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.

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 in pixels, relative to the spacing that already exists. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Use gpr_$inq_horizontal_spacing to retrieve a font's horizontal spacing.

The initial horizontal spacing is defined in the font file.

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.

Horizontal spacing is the space between each character in a string.

gpr_$set_imaging_format - Sets the imaging format of the color display.

FORMAT

```
gpr_$set_imaging_format (format, status)
```

INPUT PARAMETERS

format
Color format in gpr_$imaging_format_t. This is a two-byte integer. Valid values are:

gpr_$interactive either two- or three-board

gpr_$imaging_1024x1024x8
two-board only

gpr_$imaging_512x512x24
three-board only

OUTPUT PARAMETERS

status
Completion status, in status_$t format.

USAGE

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.

Imaging formats support only limited GPR operations - displaying pixel data and changing the color map. Other functions return error messages.

1024x1024x8 imaging format is not supported on a three-board system because it offers no advantages over interactive formats.

The only models that accept the gpr_$imaging formats are the DN550, DN560, DN600, and DN660. It is unlikely that future models of Apollo nodes will support these imaging 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.

USAGE

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_count, pattern, length, status)
```

INPUT PARAMETERS

repeat_count
: The replication factor for each bit in the pattern. This is a 2-byte integer. Specifying a value of 0 results in a solid line.

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. Specifying a value of 0 results in a solid line.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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_$set_draw_pattern returns an error if called after gpr_$set_line_pattern or gpr_$set_linestyle.

Use gpr_$set_draw_pattern to set a draw pattern for all line curve primitives including arcs. gpr_$set_line_pattern sets a line pattern only for lines and splines.

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. This is a 2-byte integer. Specify only one of the following values:

gpr_$solid for solid lines.

gpr_$dotted for dashed lines.

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.

USAGE

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_opt_t format, the action to be taken. This is a 2-byte integer. Specify only one of the following values:

  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.

  gpr_$input_ok_if_obs
  : blocks display acquisitions, but allows input into the window even if the window is obscured.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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.

Use gpr_$inq_vis_list to retrieve a list of visible sections of an obscured window.

When a program specifies gpr_$ok_if_obs, the output is performed even when the window is obscured. To avoid overwriting other Display Manager windows, the program must inquire the visible areas by calling gpr_$inq_vis_list and set clipping windows accordingly.

When a program specifies gpr_$input_ok_if_obs, the input is performed even when the window is obscured.

The cursor state (cursor pattern and whether the cursor is active) is in effect at all times, even when the display is not acquired. Three exceptions are when the window is an icon, when the window is in hold mode, and when the window is obscured and gpr_$set_obscured_opt does not specify gpr_$input_ok_if_obs.

Setting if_obscured to gpr_$block_if_obs or gpr_$ok_if_obs has an effect on the refresh procedure specified by gpr_$set_refresh_entry.

Setting if_obscured to gpr_$block_if_obs causes only the hidden display memory refresh routine to be called.

Setting if_obscured to gpr_$ok_if_obs causes both the hidden display memory and display memory refresh routines to be called.

gpr_$set_plane_mask - Establishes a 16-bit 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. This is a two-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

We recommend that you not use gpr_$set_plane_mask routine; we may obsolete this call. Please use the gpr_$set_plane_mask_32 routine instead.

gpr_$set_plane_mask_32 - Establishes a 32-bit plane mask for subsequent write operations.

FORMAT

```
gpr_$set_plane_mask_32 (mask, status)
```

INPUT PARAMETERS

mask
: The plane mask, which specifies which planes to use, in gpr_$mask_32_t format. This is a four-byte integer.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

This call is identical to the gpr_$set_plane_mask call except that gpr_$set_plane_mask_32 establishes a 32-bit mask instead of a 16-bit mask. We recommend that you use gpr_$set_plane_mask_32 instead of gpr_$set_plane_mask since we may obsolete gpr_$set_plane_mask.

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 an 8-bit pixel 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.

gpr_$set_raster_op - Specifies a raster operation for the primitives established with gpr_$raster_op_prim_set.

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.

USAGE

Use gpr_$inq_raster_ops to retrieve the current raster operation for the primitives which are specified by gpr_$raster_op_prim_set.

The default raster operation for all primitives is 3.

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. This is a pointer to a procedure.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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.

Callback of refresh routines are affected by your obscured option. See gpr_$set_obscured_opt for more information.

- 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.

gpr_$set_space_size - Specifies the amount of horizontal space that GPR should leave blank when printing a character not defined in the current 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
: Space size is the number of pixels to skip in the horizontal direction when you include a character that is not in the font. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

To retrieve a font's space size, use gpr_$inq_space_size.

The initial character widths are defined in the font file.

To use routines that 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 write a character that is not in the font. Space size is not the size of the space character. To set the size of the space character use gpr_$set_char_width.

gpr_$set_text_background_value - Specifies the color to use for text background.

FORMAT

```
gpr_$set_text_background_value (color, status)
```

INPUT PARAMETERS

color

The color to be 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	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.
-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 sets the text background color equal to the bitmap background color. For borrowed displays and memory bitmaps, the bitmap 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.

USAGE

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.

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.

USAGE

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. This is a 2-byte integer. Specify only one of the following values:

  gpr_$up

  gpr_$down

  gpr_$left

  gpr_$right

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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 to use for writing text.

FORMAT

```
gpr_$set_text_value (color, status)
```

INPUT PARAMETERS

color

The color to be used for writing text, in gpr_$pixel_value_t format. This is a 4-byte integer. The valid values are listed below:

0 - 1	For monochromatic displays.
0 - 15	An index into a 4-plane color table.
0 - 255	An index into an 8-plane color table.
0 - 16,777,215	A color value for a 24-plane true-color program.

OUTPUT PARAMETERS

status

Completion status, in status_$t format.

USAGE

To retrieve the current text value, use gpr_$inq_text_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.

gpr_$set_triangle_fill_criteria - Sets the filling criteria used with polygons that are decomposed into triangles before being rendered or polygons that are rendered directly (decomposition technique set to render exact).

FORMAT

```
gpr_$set_triangle_fill_criteria(fill_crit, status)
```

INPUT PARAMETERS

fill_crit
: The filling criteria in gpr_$triangle_fill_criteria_t format. This data type is 4 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

This call allows you to choose how polygons decomposed into triangles or polygons that are rendered without being decomposed (decomposition technique set to render exact) are filled.

Use gpr_$pgon_decomp_technique to choose a mode which controls the algorithm used to decompose polygons into trapezoids or non-overlapping triangles.

For full details on decomposition techniques, see the *Programming With Domain Graphics Primitives* manual.

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 bitmap's window. This is a character variable.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. 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.

USAGE

gpr_$spline_cubic_p draws a smooth curve starting from the current position, through each of the specified points. The current position cannot be equal to any point specified by the call.

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. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. 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.

USAGE

gpr_$spline_cubic_x draws a smooth curve starting from the current position and through each of the specified points. The x coordinate of the current position has to be less than the first x coordinate in the x array.

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. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. The values must be within the bitmap limits, unless clipping is enabled.

y
: List of the y-coordinates of all the successive positions. gpr_$coordinate_array_t, a ten-element array of 2-byte integers, is an example of such an array. The actual array can have up to 32767 elements. 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.

USAGE

gpr_$spline_cubic_y draws a smooth curve starting from the current position and through each of the specified points. The y coordinate of the current position has to be less than the first y coordinate in the y array.

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.

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.

USAGE

gpr_$start_pgon defines the first point in a polygon boundary. This routine is used in conjunction with gpr_$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, by calling one of the following:

- gpr_$close_fill_pgon
- a combination of gpr_$close_return_pgon and gpr_$multitrapezoid.
- a combination of gpr_$close_return_pgon_tri and gpr_$multitriangle.

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.

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.

USAGE

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.

No GPR information is valid after calling gpr_$terminate.

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 up to 256 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.

USAGE

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. The size of the space is set by gpr_$set_space_size.

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, excluding descenders of the character.

Upon completion of the gpr_$text routine, the current position is updated to the coordinate position where the next character would be written. This is true even if the string is partly or completely clipped. However, the current position always remains within the boundaries of the bitmap.

Note that gpr_$text can only print character strings. If you want other kinds of data (e.g., numbers) printed, you must convert the data to a character string before calling gpr_$text.

gpr_$trapezoid - Draws and fills a trapezoid.

FORMAT

```
gpr_$trapezoid (trapezoid, status)
```

INPUT PARAMETERS

trapezoid
: Trapezoid in gpr_$trap_t format.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

gpr_$trapezoid fills in a trapezoid with the color/intensity value specified with gpr_$set_fill_value or the pattern set by gpr_$set_fill_pattern. To retrieve the current fill value, use gpr_$inq_fill_value.

The GPR routines define a trapezoid as a quadrilateral with two horizontally parallel sides.

To draw an unfilled trapezoid use gpr_$polyline.

Different decomposition techniques offer different rasterizations of polygons. For details, see the *Programming With Domain Graphics Primitives* manual.

gpr_$triangle - Draws and fills a triangle.

FORMAT

```
gpr_$triangle (vertex_1, vertex_2, vertex_3, status)
```

INPUT PARAMETERS

vertex_1
: First vertex of the triangle, in gpr_$position_t format.

vertex_2
: Second vertex of the triangle, in gpr_$position_t format.

vertex_3
: Third vertex of the triangle, in gpr_$position_t format.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

gpr_$triangle fills in a triangle with the color/intensity value specified with gpr_$set_fill_value or the fill pattern set by gpr_$set_fill_pattern.

To retrieve the current fill value, use gpr_$inq_fill_value.

Different decomposition techniques offer different rasterizations of polygons. For details, see the *Programming With Domain Graphics Primitives* manual.

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.

USAGE

The font_id is returned when a program loads a file with the routine gpr_$load_font_file.

In general, you will not have to call gpr_$unload_font_file even if you have already loaded several fonts. See the *Programming With Domain Graphic Primitives* manual for details.

gpr_$wait_frame - Waits for the current frame refresh cycle to end before executing operations that modify the display.

FORMAT

```
gpr_$wait_frame (status)
```

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

Operations that modify the color display include block transfers and drawing and text operations.

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
: An array from which to write pixel values in gpr_$pixel_array_t format. This is an array of 4-byte integers. You can specify an array of up to 131,073 elements.

destination_window
: Rectangular section of the current bitmap into which to write the pixel values, in gpr_$window_t format. This data type is 8 bytes long.

OUTPUT PARAMETERS

status
: Completion status, in status_$t format.

USAGE

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.

A program cannot use this routine on a bitmap corresponding to a Display Manager frame.

Chapter 3
GPR Errors

This chapter lists possible GPR errors. A brief explanation is provided with each error.

gpr_$already_initialized
: Graphics primitives are already initialized; you tried to call gpr_$init more than once.

gpr_$arc_overflow_16bit_bounds
: Distance between points on arc exceeds the allowable 16 bits of precision.

gpr_$array_not_sorted
: Array must be in ascending order.

gpr_$bad_attribute_block
: The attribute block descriptor is incorrect.

gpr_$bad_bitmap
: The bitmap descriptor is incorrect.

gpr_$bad_decomp_tech
: Invalid decomposition technique.

gpr_$bad_font_file
: Font file is incorrect.

gpr_$bitmap_is_read_only
: Bitmap is read-only and you have tried to write to it. You may want to change the access parameter on the gpr_$open_bitmap_file to gpr_$update or GPR_$WRITE.

gpr_$bitmap_not_a_file_bitmap
: Attempting to set or inquire a bitmap file color map when you have not passed a bitmap descriptor to an external bitmap.

gpr_$cant_deallocate
: You cannot deallocate this bitmap.

gpr_$cant_mix_modes
: You cannot mix display modes; for example, borrow and direct.

gpr_$character_not_in_font
: Character is not in a font.

gpr_$coord_out_of_bounds
: Coordinate value is out of bounds.

gpr_$dest_out_of_bounds
: Destination window origin is out of bitmap bounds.

gpr_$dimension_too_big
: The bitmap dimension is too big.

gpr_$dimension_too_small
: The bitmap dimension is too small.

gpr_$display_not_acq
: Display has not been acquired. You can acquire the display with the gpr_$acquire_display routine.

gpr_$duplicate_points
: Duplicate points are illegal.

gpr_$empty_rop_prim_set
: Raster operation primitive set is empty.

gpr_$font_table_full
: Font table is full.

gpr_$font_is_read_only
: The following calls cannot be used to modify a read-only font: gpr_$set_space_size, gpr_$set_horizontal_spacing, and gpr_$set_character_width.

gpr_$illegal_fill_pattern
: Illegal bitmap for a fill pattern.

gpr_$illegal_fill_scale
: Fill pattern scale must be one.

gpr_$illegal_for_frame
: Operation is illegal for DM frame.

gpr_$illegal_for_pixel_bitmap
: You cannot call a GPR draw, fill, or text routine in a pixel-oriented bitmap.

gpr_$illegal_pattern_length
: The length of a line pattern must be less than 64 and greater than 0.

gpr_$illegal_pixel_value
: Pixel value range is illegal.

gpr_$illegal_software_version
: Pad is not compatible with current software version.

gpr_$illegal_text_path
: Value is not in gpr_$direction_t.

gpr_$illegal_when_imaging
: Operation is illegal in imaging format.

gpr_$incorrect_alignment
: Bitmap layout specifications do not satisfy GPR alignment constraints.

gpr_$incorrect_decomp_tech
: Attempting to close a polygon and return a set of trapezoids when the decomposition technique is not set to one of the trapezoid techniques or attempting to close a polygon and return a set of triangles when the decomposition technique is not set to non_overlapping_tris.

gpr_$input_buffer_overflow
: The system has discarded some input events because the input buffer overflowed. The input buffer can currently hold 256 events. To avoid this problem, your program should read the input buffer more frequently.

gpr_$internal_error
: This is an internal error.

gpr_$invalid_color_map
The color map is invalid.

gpr_$invalid_font_id
Font id is invalid.

gpr_$invalid_imaging_format
Format is invalid for display hardware.

gpr_$invalid_plane
The plane number is invalid.

gpr_$invalid_raster_op
The raster operation value is invalid.

gpr_$invalid_virtual_device_id
Invalid virtual device identification number.

gpr_$kbd_not_acq
Keyboard has not been acquired.

gpr_$must_borrow_display
You must borrow the display for this operation.

gpr_$must_have_display
Display must be acquired.

gpr_$must_release_display
You must release the display for this operation.

gpr_$no_attributes_defined
No attributes are defined for the bitmap.

gpr_$no_color_map_in_file
Attempting to inquire a bitmap file color map when you have not passed a bitmap descriptor to an external bitmap.

gpr_$no_input_enabled
No input events are enabled.

gpr_$no_more_fast_buffers
You cannot allocate another buffer bitmap.

gpr_$no_more_space
No more bitmap space is available.

gpr_$no_reset_decomp_in_pgon
Cannot set the decomposition technique between gpr_$start_pgon and gpr_$close_return_pgon, gpr_$close_fill_pgon, or gpr_$close_return_pgon_tri.

gpr_$not_in_direct_mode
Display is not in direct mode.

gpr_$not_in_polygon
No polygon is being defined.

gpr_$not_initialized
Primitives are not initialized.

gpr_$rop_sets_not_equal
: Raster operations sets are not equal.

gpr_$source_out_of_bounds
: Source window origin is out of bitmap bounds.

gpr_$specific_nonzero_only
: Must specify a winding number when the fill criterion is gpr_$specific.

gpr_$style_call_not_active
: You cannot call gpr_$set_linestyle if you have previously called gpr_$set_draw_pattern.

gpr_$unable_to_rotate_font
: Rotated character cannot fit into allocated character space.

gpr_$window_obscured
: Window is obscured.

gpr_$window_out_of_bounds
: Window origin is out of bitmap bounds.

gpr_$wrong_display_hardware
: The display hardware is wrong for this operation.

Index

Acquiring display 2-2
Arcs 2-9, 2-10
Attribute blocks 2-4
    deallocating 2-24
    inquiring 2-11
    setting 2-4, 2-104

Bitmaps
    clearing 2-16
    deallocating 2-25
    file 2-85
    hidden display memory 2-8
    inquiring 2-39
    main memory 2-5, 2-6
    number of planes 2-107
    pointers to 2-42, 2-44
    position within 2-43, 2-81
    setting current 2-106
    size of 2-107
BLT 2-3, 2-12, 2-92
Boxes 2-28

Character width 2-110
    inquiring 2-45
Circles 2-14, 2-15
    filled 2-15
    unfilled 2-14
Clearing bitmaps 2-16
Clip windows 2-111, 2-112
    inquiring 2-48
Color map 2-113
    inquiring 2-46
Conditional event waiting 2-22
Configuration
    inquiring 2-47
Constants 1-3
Current position 2-81
    inquiring 2-50
Cursor 2-116
    inquiring 2-51
    origin 2-117
    pattern 2-118
    position 2-119

Data types 1-3
Deallocating
    attribute blocks 2-24
    bitmaps 2-25
    double buffers 2-26
Decomposition techniques
    inquiring 2-65
    setting 2-89
Direct mode
    acquiring 2-2
    releasing 2-97
Disabling input 2-27
Display characteristics
    inquiring 2-53
Display memory bitmaps
    creating 2-36
    deallocating 2-25, 2-39
Display type
    inquiring 2-47
Double buffering 2-7
Double buffers
    creating 2-7
    deallocating 2-26
    inquiring 2-77
    selecting 2-102
Drawing 2-120
    color 2-56, 2-122
    pattern 2-55, 2-120
    width 2-57, 2-123

Enabling direct access 2-29
Enabling input 2-30
Event counting 2-35
Event enabling 2-30
Event reporting 2-22, 2-32
Event waiting 2-32
External file bitmaps 2-85

File bitmaps 2-85
File color map 2-109
Fill background 2-124
    inquiring 2-58
Fill color 2-126
    inquiring 2-60

Fill pattern
    inquiring 2-59
Fill patterns 2-125
Font files 2-80, 2-154
Frame 0 2-101
Frame 1 2-101
Frame refresh 2-155

Gpr_$acquire_display 2-2
Gpr_$additive_blt 2-3
Gpr_$allocate_attribute_block 2-4
Gpr_$allocate_bitmap 2-5
Gpr_$allocate_bitmap_nc 2-6
Gpr_$allocate_buffer 2-7
Gpr_$allocate_hdm_bitmap 2-8
Gpr_$arc_3p 2-10
Gpr_$arc_c2p 2-9
Gpr_$attribute_block 2-11
Gpr_$bit_blt 2-12
Gpr_$circle 2-14
Gpr_$circle_filled 2-15
Gpr_$clear 2-16
Gpr_$close_fill_pgon 2-17
Gpr_$close_return_pgon 2-18
Gpr_$close_return_pgon_tri 2-19
Gpr_$color_zoom 2-20
Gpr_$cond_event_wait 2-22
Gpr_$deallocate_attribute_block 2-24
Gpr_$deallocate_bitmap 2-25
Gpr_$deallocate_buffer 2-26
Gpr_$disable_input 2-27
Gpr_$draw_box 2-28
Gpr_$enable_direct_access 2-29
Gpr_$enable_input 2-30
Gpr_$event_wait 2-32
Gpr_$force_release 2-34
Gpr_$get_ec 2-35
Gpr_$init 2-36
Gpr_$inq_bitmap 2-39
Gpr_$inq_bitmap_dimensions 2-40
Gpr_$inq_bitmap_file_color_map 2-41
Gpr_$inq_bitmap_pointer 2-42
Gpr_$inq_bitmap_position 2-43
Gpr_$inq_bm_bit_offset 2-44
Gpr_$inq_character_width 2-45
Gpr_$inq_color_map 2-46
Gpr_$inq_config 2-47
Gpr_$inq_constraints 2-48
Gpr_$inq_coordinate_origin 2-49
Gpr_$inq_cp 2-50
Gpr_$inq_cursor 2-51
Gpr_$inq_disp_characteristics 2-53
Gpr_$inq_draw_pattern 2-55
Gpr_$inq_draw_value 2-56
Gpr_$inq_draw_width 2-57
Gpr_$inq_fill_background_value 2-58
Gpr_$inq_fill_pattern 2-59
Gpr_$inq_fill_value 2-60
Gpr_$inq_horizontal_spacing 2-61
Gpr_$inq_imaging_format 2-62
Gpr_$inq_line_pattern 2-63
Gpr_$inq_linestyle 2-64
Gpr_$inq_pgon_decomp_technique 2-65
Gpr_$inq_raster_op_prim_set 2-66
Gpr_$inq_raster_ops 2-67
Gpr_$inq_refresh_entry 2-68
Gpr_$inq_space_size 2-69
Gpr_$inq_text 2-70
Gpr_$inq_text_extent 2-71
Gpr_$inq_text_offset 2-72
Gpr_$inq_text_path 2-73
Gpr_$inq_text_values 2-74
Gpr_$inq_triangle_fill_criteria 2-75
Gpr_$inq_vis_list 2-76
Gpr_$inq_visible_buffer 2-77
Gpr_$inq_window_id 2-78
Gpr_$line 2-79
Gpr_$load_font_file 2-80
Gpr_$move 2-81
Gpr_$multiline 2-82
Gpr_$multitrapezoid 2-83
Gpr_$multitriangle 2-84
Gpr_$open_bitmap_file 2-85
Gpr_$pgon_decomp_technique 2-89
Gpr_$pgon_polyline 2-91
Gpr_$pixel_blt 2-92
Gpr_$polyline 2-93
Gpr_$raster_op_prim_set 2-94
Gpr_$read_pixels 2-95

Gpr_$rectangle 2-96
Gpr_$release_display 2-97
Gpr_$remap_color_memory 2-98
Gpr_$remap_color_memory_1 2-99
Gpr_$replicate_font 2-100
Gpr_$select_color_frame 2-101
Gpr_$select_display_buffer 2-102
Gpr_$set_attribute_block 2-104
Gpr_$set_auto_refresh 2-105
Gpr_$set_bitmap 2-106
Gpr_$set_bitmap_dimensions 2-107
Gpr_$set_bitmap_file_color_map 2-109
Gpr_$set_character_width 2-110
Gpr_$set_clip_window 2-111
Gpr_$set_clipping_active 2-112
Gpr_$set_color_map 2-113
Gpr_$set_coordinate_origin 2-115
Gpr_$set_cursor_active 2-116
Gpr_$set_cursor_origin 2-117
Gpr_$set_cursor_pattern 2-118
Gpr_$set_cursor_position 2-119
Gpr_$set_draw_pattern 2-120
Gpr_$set_draw_value 2-122
Gpr_$set_draw_width 2-123
Gpr_$set_fill_background_value 2-124
Gpr_$set_fill_pattern 2-125
Gpr_$set_fill_value 2-126
Gpr_$set_horizontal_spacing 2-127
Gpr_$set_imaging_format 2-128
Gpr_$set_input_sid 2-129
Gpr_$set_line_pattern 2-130
Gpr_$set_linestyle 2-131
Gpr_$set_obscured_opt 2-132
Gpr_$set_plane_mask 2-134
Gpr_$set_plane_mask_32 2-135
Gpr_$set_raster_op 2-136
Gpr_$set_refresh_entry 2-138
Gpr_$set_space_size 2-139
Gpr_$set_text_background_value 2-140
Gpr_$set_text_font 2-141
Gpr_$set_text_path 2-142
Gpr_$set_text_value 2-143
Gpr_$set_triangle_fill_criteria 2-144
Gpr_$set_window_id 2-145
Gpr_$spline_cubic_p 2-146
Gpr_$spline_cubic_x 2-147
Gpr_$spline_cubic_y 2-148
Gpr_$start_pgon 2-149
Gpr_$terminate 2-150
Gpr_$text 2-151
Gpr_$trapezoid 2-152
Gpr_$triangle 2-153
Gpr_$unload_font_file 2-154
Gpr_$wait_frame 2-155
Gpr_$write_pixels 2-156

Hidden display memory bitmaps 2-8
    deallocating 2-25, 2-39
Horizontal spacing 2-127
    inquiring 2-61

Imaging mode 2-128
    inquiring 2-62
Initializing GPR 2-36
Input disabling 2-27
Input enabling 2-30
Input pad 2-129
Inquiring 2-40
    bitmap addresses 2-44
    bitmap pointers 2-42
    bitmap position 2-43
    character size 2-69
    clipping 2-48
    color map 2-46
    configuration 2-47
    current position 2-50
    cursor 2-51
    decomposition techniques 2-65
    display characteristics 2-53
    draw color 2-56
    draw pattern 2-55
    draw width 2-57
    file color map 2-41
    fill background 2-58
    fill color 2-60
    fill pattern 2-59
    horizontal spacing 2-61
    imaging mode 2-62
    line pattern 2-63
    linestyle 2-64
    number of planes 2-53
    obscured windows 2-76
    origin 2-49
    plane masks 2-48

raster operations 2-66, 2-67
refresh procedures 2-68
text color 2-74
text extent 2-71
text font 2-70
text offset 2-72
text path 2-73
tile fills 2-58
triangle fill criteria 2-75
width of characters 2-45
window id 2-78

Line patterns 2-130
inquiring 2-63
Lines 2-79, 2-82, 2-93
Linestyle 2-131
inquiring 2-64
Loading font files 2-80

Magnifying 2-20
Main memory bitmaps 2-5, 2-6
creating 2-36
deallocating 2-25, 2-39

Origin 2-115
inquiring 2-49

Plane masks 2-135
inquiring 2-48
Planes 2-36
inquiring 2-53
Polygons 2-91
closing 2-18, 2-19
filling 2-17

Raster operations
inquiring 2-66, 2-67
Reading pixels 2-95
Rectangles 2-96
Refresh procedures 2-138
inquiring 2-68
setting 2-105
Releasing display 2-34, 2-97
Remapping color memory 2-98, 2-99

Setting
auto-refresh 2-105
bitmap position 2-81
decomposition techniques 2-89
Stopping GPR 2-150

Terminating GPR 2-150
Text
inquiring about color 2-74
inquiring about extent 2-71
inquiring about font 2-70
inquiring about offset 2-72
inquiring about path 2-73
loading font files 2-80
unloading font files 2-154
Tile fills 2-124
inquiring 2-58
Trapezoids 2-83, 2-152
Triangle fill criteria
inquiring 2-75
Triangles 2-84, 2-153

Unloading font files 2-154

Waiting for refresh 2-155
Windows
id 2-78
visible sections 2-76
Writing pixels 2-156

Zooming 2-20

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Domain Graphics Primitives Resource Call Reference*
Order No.: 007194 Revision: 02 Date of Publication: June, 1987

What type of user are you?

_____ System programmer; language _____
_____ Applications programmer; language _____
_____ System maintenance person
_____ System Administrator
_____ Student Programmer
_____ Other

_____ Manager/Professional
_____ Technical Professional
_____ Novice

How often do you use the DOMAIN system?_____

What parts of the manual are especially useful for the job you are doing?_____

What additional information would you like the manual to include?_____

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible. Specify additional index entries.)_____

_____
Your Name Date

_____
Organization

_____
Street Address

_____
City State Zip

No postage necessary if mailed in the U.S.

cut or fold along dotted line

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

FOLD