DOMAIN Graphics Primitive Resource Call Reference

Order No. 007194
Revision 01

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

Printed in U.S.A.

First Printing: January, 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 the 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	Presents a description of each routine including format and parameters. The organization of routines is alphabetical.
Chapter 3	Presents a GPR error listing.

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 01) 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 DOMAIN Graphics Primitives.

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 User's Guide* (000792), and the *DOMAIN C Language Reference* (002093).

Documentation Conventions

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

UPPERCASE	Uppercase words or characters in formats and command descriptions represent commands or keywords that you must use literally.
lowercase	Lowercase words or characters in formats and command descriptions represent values that you must supply.
[ ]	Square brackets enclose optional items in formats and command descriptions. In sample Pascal statements, square brackets assume their Pascal meanings.
{ }	Braces enclose a list from which you must choose an item in formats and command descriptions. In sample Pascal statements, braces assume their Pascal meanings.
CTRL/Z	The notation CTRL/ followed by the name of a key indicates a control character sequence. You should hold down the <CTRL> key while typing the character.
	Vertical ellipses represent additional information in a program fragment that is either too lengthy to include or not relevant to the example.

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.

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). Each data type description includes an atomic data type translation (i.e., GPR_$LINESTYLE_T = 2-byte integer) as well as a brief description of the type's purpose. The description includes any predefined values associated with the type. The following is an example of a data type description for the GPR_$LINESTYLE_T type:

GPR_$LINESTYLE_T

A 2-byte integer. Specifies the linestyle for line-drawing operations One of the following predefined values:

GPR_$SOLID
Draw solid lines.

GPR_$DOTTED
Draw dotted lines.

This chapter also illustrates the record data types in detail. These illustrations will help FORTRAN programmers construct record-like structures, as well as provide useful information for all programmers. Each record type illustration:

- Shows FORTRAN programmers the structure of the record that they must construct using standard FORTRAN data-type statements. The illustrations show the size and type of each field.
- Describes the fields that make up the record.
- Lists the byte offsets for each field. Use these offsets to access individual fields. Bytes are numbered from left to right and bits are numbered from right to left.
- Indicates whether any fields of the record are, in turn, predefined records.

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_$DEFAULT_LIST_SIZE	10	
GPR_$GREEN	16#00FF00	color value for green
GPR_$HIGHEST_PLANE	7	max plane number in a bitmap
GPR_$MAGENTA	16#FF00FF	color value for magenta (red + blue)
GPR_$MAX_BMF_GROUP	0	max group in external bitmaps
GPR_$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_$STRING_SIZE	256	number of chars in a gpr string
GPR_$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)
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

DATA TYPES

GPR_$ACCELERATOR_TYPE_T

A 2-byte integer. Unique number corresponding to the graphics accelerator processor type One of the following predefined values:

GPR_$ACCEL_NONE
None or not applicable.

GPR_$ACCEL_1
3DGA.

GPR_$ACCESS_ALLOCATION_T

A 2-byte integer. The legal allocated sizes of pixel cells in bitmap sections for direct access. 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

A 2-byte integer. The ways to access an external bitmap. 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

A 2-byte integer. The set of legal allocated sizes of pixel cells in bitmap sections for direct access.

GPR_$ATTRIBUTE_DESC_T

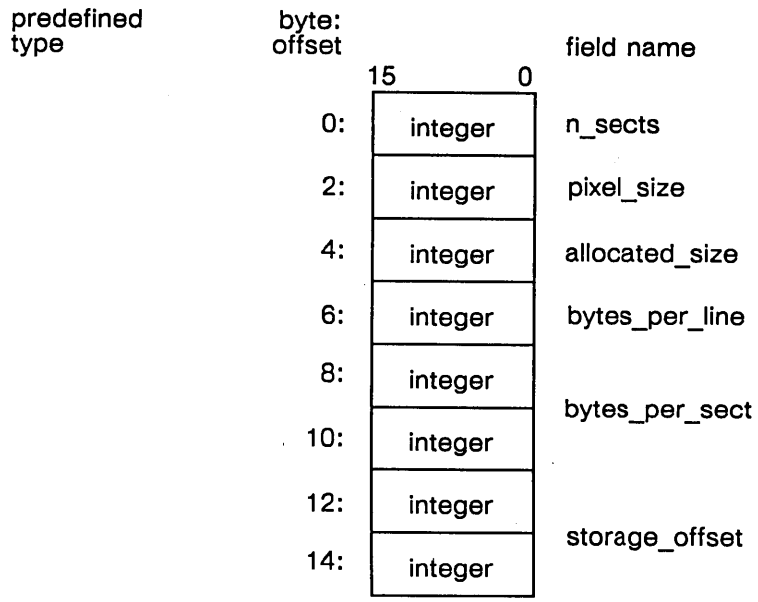
A 4-byte integer. Identifies an attribute block.

GPR_$BITMAP_DESC_T

A 4-byte integer. Identifies a bitmap.

GPR_$BMF_GROUP_HEADER_T

The group header description for an external bitmap. The diagram below illustrates the GPR_$BMF_GROUP_HEADER_T data type:



Field Description:

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

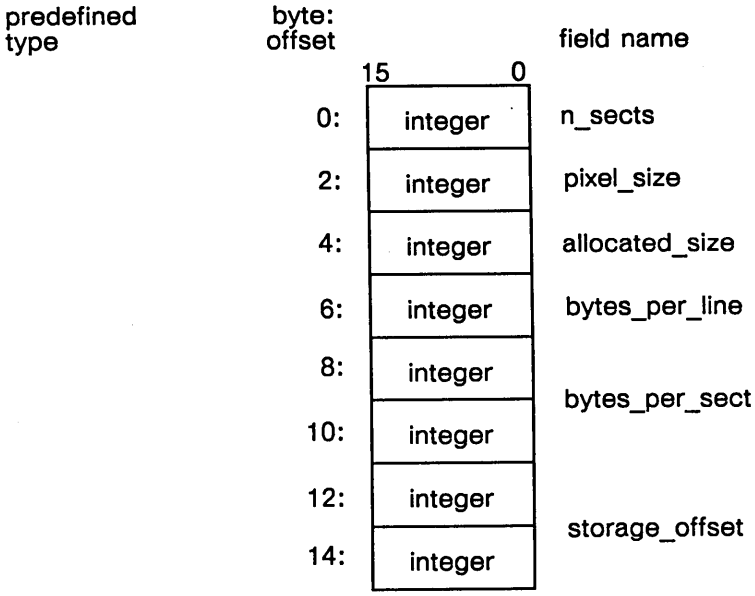
bytes_per_line
The number of bytes in one row of a bitmap.

bytes_per_sect
The number 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

A gpr_$max_bmf_group-element array of gpr_bmf_group_header_t record structures. The diagram below illustrates a single element:



Field Description:

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

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

bytes_per_sect
The number 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_$COLOR_T

A 4-byte integer. Defines a color.

GPR_$COLOR_VECTOR_T

A 256-element array of 4-byte integers. Stores multiple color values. Arrays of this type are used as input parameters of color values to be inserted into consecutive slots of a color map. They are also used as output parameters to store color values when inquiries are performed on color maps.

GPR_$CONTROLLER_TYPE_T

A 2-byte integer. Unique number corresponding to the display controller type. 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

GPR_$CTL_COLOR_3
DN570/570A

GPR_$CTL_COLOR_4
DN3000

GPR_$CTL_MONO_4
DN3000

GPR_$COORDINATE_ARRAY_T

A 10-element array of 2-byte integers. Specifies several coordinates in a bitmap. Generally, x coordinates are passed in one array and y coordinates are passed in another array.

GPR_$COORDINATE_T

A 2-byte integer. Specfies one coordinate in a bitmap.

GPR_$DECOMP_TECHNIQUE_T

A 2-byte integer. Specifies a decomposition technique. 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

A 2-byte integer. Specifies the direction of movement from one text character position to another in a bitmap. One of the following predefined values:

GPR_$UP

GPR_$DOWN

GPR_$LEFT

GPR_$RIGHT

GPR_$DISP_CHAR_T

Stores display characteristics. The diagram below illustrates the gpr_$disp_char_t data type:

predefined type	byte: offset	15 0	field name
gpr_$controller_type_t	0:	integer	controller_type
gpr_$accelerator_type_t	2:	integer	accelerator_type
	4:	integer	x_window_origin
	6:	integer	y_window_origin
	8:	integer	x_window_size
	10:	integer	y_window_size
	12:	integer	x_visible_size
	14:	integer	y_visible_size
	16:	integer	x_extension_size
	18:	integer	y_extension_size
	20:	integer	x_total_size
	22:	integer	y_total_size
	24:	integer	x_pixels_per_cm
	26:	integer	y_pixels_per_cm

predefined type	byte: offset		field name
	28:	integer	n_planes
	30:	integer	n_buffers
	32:	integer	delta_x_per_buffer
	34:	integer	delta_y_per_buffer
	36:	integer	delta_planes_per_buffer
gpr_$overlap_set_t	38:	integer	mem_overlaps
	40:	integer	x_zoom_max
	42:	integer	y_zoom_min
	44:	integer	video_refresh_rate
	46:	integer	n_primaries
	48:	integer	lut_width_per_primary
gpr_$format_set_t	50:	integer	avail_formats
gpr_$access_set_t	52:	integer	avail_access
	54:	integer	access_address_space
gpr_disp_invert_t	56:	integer	invert

Field Description:

CONTROLLER_TYPE

A 2-byte integer. The type of graphics hardware controller. 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
DN600/550/560

GPR_$CTL_COLOR_2
580

GPR_$CTL_COLOR_3
DN570

GPR_$CTL_COLOR_4
DN3000 color.

```
For gpr_$no_display mode, gpr_$ctl_none is returned.
Note that code which makes use of these values may not automatically
extend to new node types, since as new controllers are released,
they will be given new values, and this list will be extended.
```

ACCELERATOR_TYPE — A 2-byte integer. The type of graphics hardware processing accelerator for the node. Only one of the following values is returned. One of the following predefined values:

GPR_$ACCEL_NONE
none or not applicable.

NOTE:

```
Code which makes use of these values may not
automatically extend to new node types, since as
new controllers are released, they will be given
new values, and this list will be extended.

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
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.

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

A 2-byte integer. This parameter gives the kinds of overlap situations that can exist between refresh buffer memory that may be used for different purposes in the device. 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. Any combination 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.

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 which 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 which 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 — A 2-byte integer. The set of available interactive or imaging formats available on the device. Any combination of the following predefined values:

GPR_$INTERACTIVE
Interactive format

GPR_$IMAGING_1024X1024X8
8-bit pixel format on a two-board configuration

GPR_$IMAGING_512X512X24
24-bit pixel format on a three-board configuration

AVAIL_ACCESS — A 2-byte integer. 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. Any combination 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

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

A 2-byte integer. This parameter is intended for monochromatic devices. It indicates how the display manager's INV is implemented on the device. One of the following predefined values:

GPR_$ACCEL_NONE
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

A 2-byte integer. Specifies the hardware configuration. 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 four-plane color display.

GPR_$COLOR_1024X1024X8
An eight-plane color display.

GPR_$COLOR_1024X800X4
An four-plane color display.

GPR_$COLOR_1024X800X8
An eight-plane color display.

GPR_$COLOR_1280X1024X8
Two-board, eight-plane display.

GPR_$COLOR1_1024X800X8
Two-board, eight-plane display.

GPR_$COLOR2_1024X800X4
One-board, four-plane display.

GPR_$BW_1280X1024
Black and white display.

GPR_$DISPLAY_INVERT_T

A 2-byte integer. The different color map implementations on monochromatic displays. One of the following predefined values:

GPR_$NO_INVERT
Not applicable, that is, a color monitor or no display.

GPR_$INVERT_SIMULATE
The color map is simulated in software.

GPR_$INVERT_HARDWARE
The color map is in hardware.

GPR_$DISPLAY_MODE_T

A 2-byte integer. Specifies the mode of operation. One of the following predefined values:

GPR_$BORROW
Uses the entire screen.

GPR_$FRAME
Uses a frame of the Display Manager.

GPR_$NO_DISPLAY
Uses a main-memory bitmap.

GPR_$DIRECT
Uses a display-manager window.

GPR_$BORROW_NC
Uses the entire screen but does not clear the bitmap.

GPR_$EC_KEY_T

A 2-byte integer. GPR_$INPUT_EC is a predefined value.

GPR_$EVENT_T

A 2-byte integer. Specifies the type of input event. One of the following predefined values:

GPR_$KEYSTROKE
When keyboard character is typed.

GPR_$BUTTONS
When you press 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.

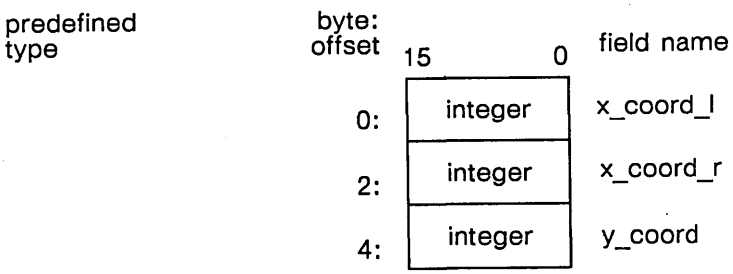
GPR_$NO_EVENT

GPR_$FORMAT_SET_T

A 2-byte integer. Specifies a set of imaging formats.

GPR_$HORIZ_SEG_T

Defines the left- and right-hand x coordinates and the y coordinate of a horizontal line segment. The diagram below illustrates the gpr_$horiz_seg_t data type:



Field Description:

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

A 2-byte integer. Specifies an imaging or interactive display format. 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

An 8-element array of 4-byte integers. Specifies the set of characters that make up a keyset associated with the graphics input event types GPR_$KEYSTROKE and GPR_$BUTTONS. The maximum number of elements in a keyset is 256. Each element of the set is represented by one bit.

GPR_$LINE_PATTERN_T

A 4-element array of 2-byte integers. Specifies the line-pattern to use for line-drawing operations

GPR_$LINESTYLE_T

A 2-byte integer. Specifies the linestyle for line-drawing operations One of the following predefined values:

GPR_$SOLID
Draw solid lines.

GPR_$DOTTED
Draw dotted lines.

GPR_$MASK_T

A 2-byte integer. Specifies a set of planes to be used in a plane mask.

GPR_$MEMORY_OVERLAP_T

A 2-byte integer. Kinds of memory overlaps between different classes of buffer memory. 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

A 2-byte integer. Specifies the action when a window is obscured. 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 allows input into the window even though the window is obscured.

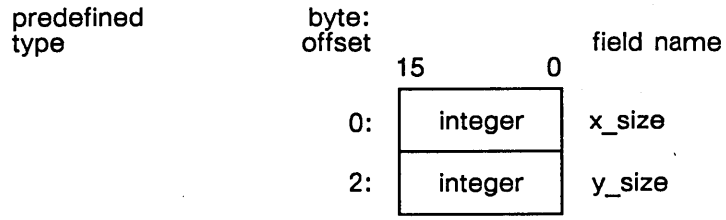
GPR_$ERROR_IF_OBS
Do not acquire the display; return an error message.

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

Specifies the width and height of a window. The diagram below illustrates the gpr_$offset_t data type:



Field Description:

x_size
The width of the window in pixels.

y_size
The height of the window in pixels.

GPR_$OVERLAP_SET_T

A 2-byte integer. Specifies a set of overlaps between different classes of buffer memory.

GPR_$PIXEL_ARRAY_T

A 131073-element array of 4-byte integers. Stores multiple pixel values.

GPR_$PIXEL_VALUE_T

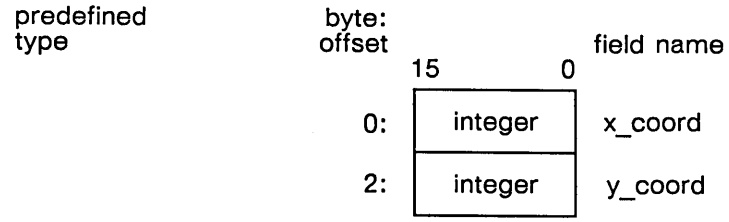
A 4-byte integer. Defines an index into a color map to identify the color of an individual pixel.

GPR_$PLANE_T

A 2-byte integer. Specifies the number of planes in a bitmap.

GPR_$POSITION_T

Specifies the x and y coordinates of a point in a bitmap. The diagram below illustrates the gpr_$position_t data type:



Field Description:

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

A 8-element array of 2-byte integers. Stores multiple raster operation opcodes

GPR_$RASTER_OP_T

A 2-byte integer. Specifies raster operation opcodes.

GPR_$ROP_PRIM_SET_ELEMS_T

A 2-byte integer. Specifies the primitives to which raster operations are applied. Any combination 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

A 2-byte integer. 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. The maximum number of elements in the set is 3. Each element of the set is represented by one bit.

GPR_$RHDM_PR_T

A 4-byte integer. A pointer to a procedure used for refresh-hidden display memory procedures.

GPR_$RWIN_PR_T

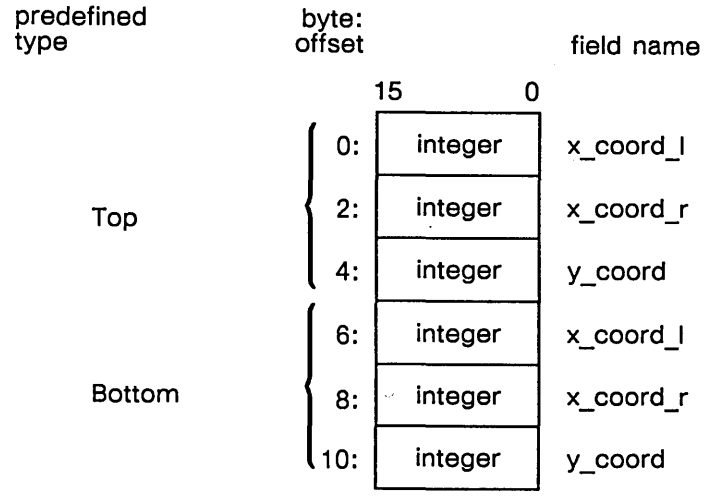
A 4-byte integer. A pointer to a procedure used for refresh-window procedures.

GPR_$STRING_T

An array of up to 256 characters. Stores up to 256 characters.

GPR_$TRAP_LIST_T

A 10-element array of gpr_$trap_t record structures. The diagram below illustrates a single element:



Field Description:

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.

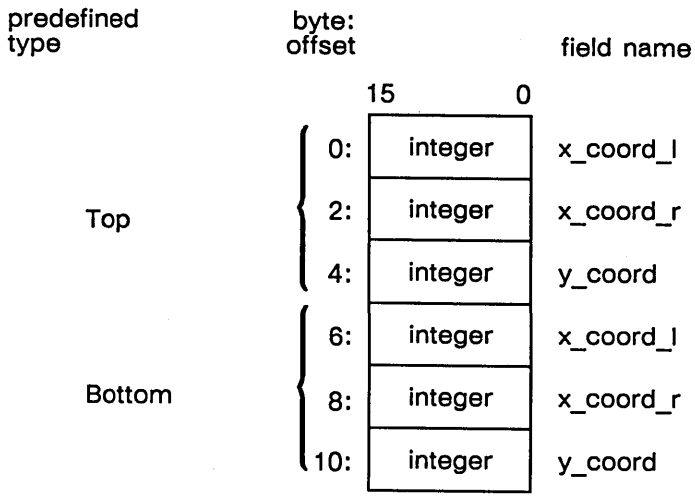
top.y_coord
The y_coordinate of the top line.
bot.x_coord_l
The left-hand x_coordinate of the bottom line.

bot.x_coord_r
The right-hand x_coordinate of the bottom line.

bot.y_coord
The y_coordinate of the bottom line.

GPR_$TRAP_T

Specifies the coordinates of the top and bottom line segments of a trapezoid. The diagram below illustrates the gpr_$trap_t data type:



Field Description:

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.

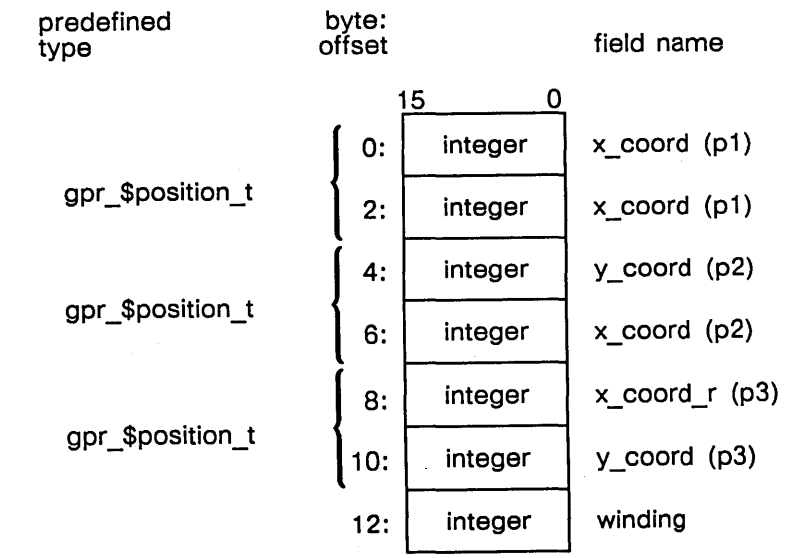
bot.x_coord_l
The left-hand x_coordinate of the bottom line.

bot.x_coord_r
The right-hand x_coordinate of the bottom line.

bot.y_coord
The y_coordinate of the bottom line.

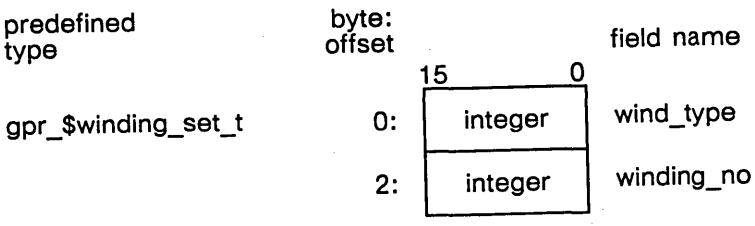
GPR_$TRIANGLE_LIST_T

A 10-element array of gpr_$triangle_t record structures. The diagram below illustrates a single element:



GPR_$TRIANGLE_FILL_CRITERIA_T

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:



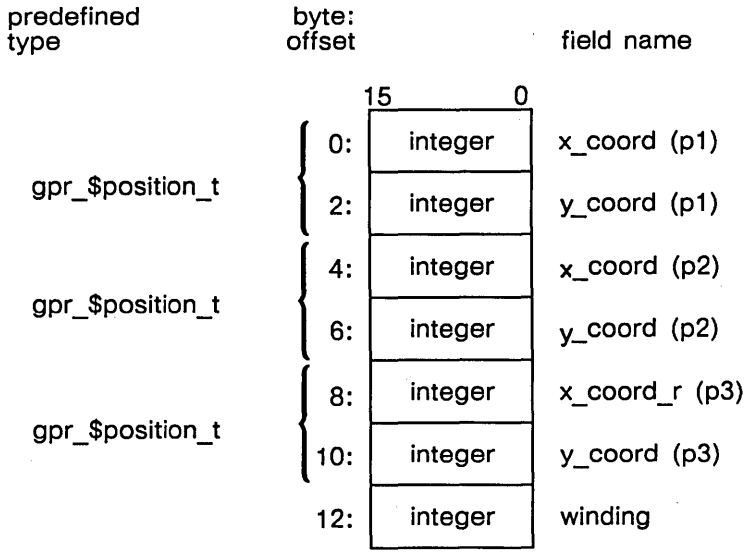
Field Description:

wind_type
The type of fill criterion to use. That is, GPR_$PARITY, GPR_$NONZERO, or GPR_$SPECIFIC.

winding_no
The winding number to be used when the wind_type is GPR_$SPECIFIC.

GPR_$TRIANGLE_T

Specifies the coordinates of a triangle. The diagram below illustrates the gpr_$triangle_t data type:



Field Description:

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.

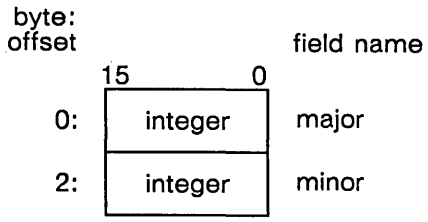
p3.x_coord_
The x coordinate of point 3.

p3.y_coord
The y coordinate of point 3.

winding
The winding number.

GPR_$VERSION_T

The version number of an external bitmap header. The diagram below illustrates the gpr_$version_t data type:



Field Description:

major
The major version number.

minor
The minor version number.

GPR_$WINDING_SET_T

A 2-byte integer. Specifies a fill criterion. One of the following predefined values:

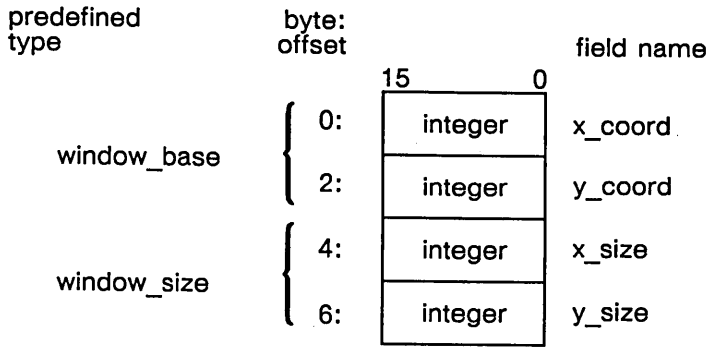
GPR_$PARITY
Apply a parity fill.

GPR_$NONZERO
Apply a nonzero fill.

GPR_$SPECIFIC
Fill areas with a specific winding number.

GPR_$WINDOW_LIST_T

A 10-element array of gpr_$window_t record structures. The diagram below illustrates a single element:



Field Description:

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

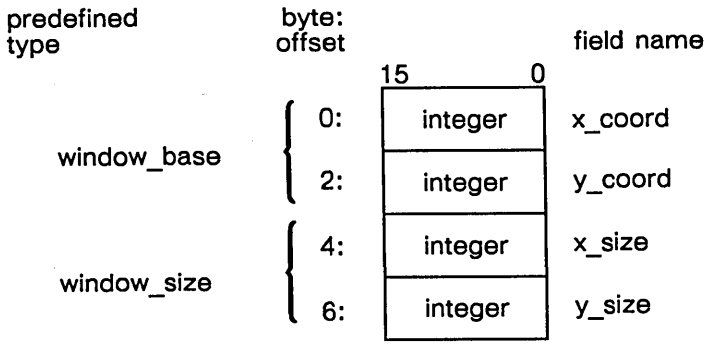
window_base.y_coord
The y coordinate of the top left-hand corner of the window.

window_size.x_size
The width of the widow in pixels.

window_size.y_size
The height of the window in pixels.

GPR_$WINDOW_T

Defines a rectanglar 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:



Field Description:

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

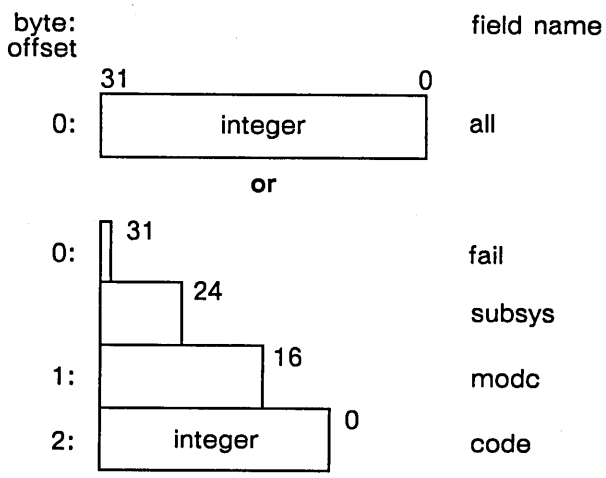
window_base.y_coord
The y coordinate of the top left-hand corner of the window.

window_size.x_size
The width of the widow in pixels.

window_size.y_size
The height of the window in pixels.

STATUS_$T

A status code. The diagram below illustrates the STATUS_$T data type:



Field Description:

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 user-callable routine descriptions alphabetically for quick reference. Each routine description contains:

- An abstract of the routine's function
- The order of the routine parameters
- A brief description of each parameter
- A description of the routine's function and use

If the parameter can be declared using a predefined data type, the description contains the phrase "in XXX format", where XXX is the predefined data type. Pascal and C programmers, look for this phrase to determine how to declare a parameter.

FORTRAN programmers, look for the phrase that describes the data type in atomic terms, such as "This parameter is a 2-byte integer." For a complete description of each data type see Chapter 1.

The rest of the parameter description describes the use of the parameter and the values it may hold.

The following is an example of a parameter description:

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
```

The GPR (Graphics Primitives) programming calls perform graphics operations within windows and window panes. This section describes their data types, call syntax, and error codes. Refer to the Introduction at the beginning of this manual for a description of data type diagrams and call syntax format.

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

USAGE

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

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

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

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

GPR_$ADDITIVE_BLT

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 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. See the GPR Data Types section for more information.

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

dest_origin
Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information. Coordinate values must be within the limits of the current bitmap, unless clipping is enabled.

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

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

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_$PLANE_T format. This is a 2-byte integer. Valid values are 0 - 7.

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

OUTPUT PARAMETERS

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

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

USAGE

To deallocate a bitmap, use GPR_$DEALLOCATE_BITMAP.

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

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

GPR_$ALLOCATE_BITMAP_NC

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

FORMAT

```
GPR_$ALLOCATE_BITMAP_NC (size,hi_plane_id,attrib_block_desc,bitmap_desc,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. See the GPR Data Types section for more information.

hi_plane_id
Identifier of the highest plane which the bitmap will use, in GPR_$PLANE_T format. This is a 2-byte integer. Valid values are 0 - 7.

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

OUTPUT PARAMETERS

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

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

USAGE

To deallocate a bitmap, use GPR_$DEALLOCATE_BITMAP.

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

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

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

GPR_$ALLOCATE_HDM_BITMAP

Allocates a bitmap in hidden display memory.

FORMAT

```
GPR_$ALLOCATE_HDM_BITMAP (size, hi_plane_id, attrib_block_desc, bitmap_desc,
                          status)
```

INPUT PARAMETERS

size
The width and height of the bitmap, in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

USAGE

GPR_$ALLOCATE_HDM_BITMAP allocates a GPR bitmap in hidden display memory for programs in borrow-display or direct mode. In frame mode, hidden display memory bitmaps cannot be used.

In direct mode you must acquire the display before calling GPR_$ALLOCATE_HDM_BITMAP.

The maximum size allowed for hidden display memory bitmaps is 224 bits by 224 bits.

Use GPR_$DEALLOCATE_BITMAP to deallocate a hidden display bitmap.

GPR_$ARC_3P

Draws an arc from the current position through two other specified points.

FORMAT

```
GPR_$ARC_3P (point_2, point_3, status)
```

INPUT PARAMETERS

point_2
The second point on the arc, in GPR_$POSITION_$T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

point_3
The third point on the arc, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

The coordinates you specify are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the arc is drawn.

After the arc is drawn, point_3 becomes the current position.

An error is returned if any of the three points are equal.

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

GPR_$ATTRIBUTE_BLOCK

Returns the descriptor of the attribute block associated with the given bitmap.

FORMAT

```
attrib_block_desc = GPR_$ATTRIBUTE_BLOCK (bitmap_desc, status)
```

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

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. See the GPR Data Types section for more information.

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

dest_origin
Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

OUTPUT PARAMETERS

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

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_$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. See the GPR Data Types section for more information.

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

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. See the GPR Data Types section for more information.

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

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/intensity value.

FORMAT

```
GPR_$CLEAR (index, status)
```

INPUT PARAMETERS

index

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

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

OUTPUT PARAMETERS

status

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

USAGE

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

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

You can use GPR_$SET_COLOR_MAP to establish the correspondence between color map indexes and color/intensity values. This means that you can use GPR_$SET_COLOR_MAP to assign the pixel value 0 to bright intensity, and then use GPR_$CLEAR either to make the screen bright by passing the pixel value 0, or make the screen dark by passing the value 1. This routine is subject to the restrictions of the current clipping window and plane mask.

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

USAGE

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

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

Filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS contain fewer pixels than filled areas rasterized with the decomposition technique set to either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS OVERLAP.

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. See GPR Data Types section for more information.

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

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.

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. See the GPR Data Types section for more information.

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

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.

GPR_$COLOR_ZOOM

Sets the zoom scale factor for a color display.

FORMAT

```
GPR_$COLOR_ZOOM (xfactor, yfactor, status)
```

INPUT PARAMETERS

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

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

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

If the x value is not equal to 1, then the y value must be not equal to 1.

GPR_$COLOR_ZOOM uses the integer zoom feature of the color hardware.

GPR_$COLOR_ZOOM works only in borrow-display mode.

GPR_$COLOR_ZOOM always zooms from the upper-left corner of the display.

GPR_$COLOR_ZOOM returns an error on models DN570/570A and DN3000 if any values other than xfactor = 1, yfactor = 1 are entered.

DN580s allow the xfactor and yfactor to be 2.

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_$SET_OBSCURED_OPT and specified an option of GPR_$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. 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. See the GPR Data Types section for more information.

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

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.

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.

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

GPR_$DEALLOCATE_ATTRIBUTE_BLOCK

Deallocates an attribute block allocated by GPR_$ALLOCATE_ATTRIBUTE_BLOCK.

FORMAT

```
GPR_$DEALLOCATE_ATTRIBUTE_BLOCK (attrib_block_desc, status)
```

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

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

USAGE

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

GPR_$DISABLE_INPUT

Disables a previously enabled event type.

FORMAT

```
GPR_$DISABLE_INPUT (event_type, status)
```

INPUT PARAMETERS

event_type

The type of event to be disabled, in GPR_$EVENT_T format. 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. This data type is 4 bytes long. See the GPR Data Types section for more information.

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-display mode, disabled events received by the GPR software will be ignored.

In direct mode or frame mode, disabled keystroke or button events are processed by the Display Manager.

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

GPR_$DRAW_BOX

Draws an unfilled box based on the coordinates of two opposing corners.

FORMAT

```
GPR_$DRAW_BOX (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. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

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 in GPR_$KEYSET_T format. In Pascal, this is a set of characters. In FORTRAN and C this can be implemented as an eight element array of 4-byte integers. This parameter is specified for event types of GPR_$KEYSTROKE and GPR_$BUTTONS. See GPR Data Types section for more information.

OUTPUT PARAMETERS

status

Completion status, in STATUS_$T format. This data type is four bytes long. See the GPR Data Types section for more information.

USAGE

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

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

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

No event types are enabled by default.

The keyset must correspond to the specified event type. For example, use ['#'..'~'] (in Pascal) to enable all normal printing graphics. Use [chr(0)..chr(127)] to enable the entire ASCII character set. Except in borrow-display mode, it is a good idea to leave at least the CMD and NEXT_WINDOW keys out of the keyset so that the user can access other Display Manager windows.

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

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

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

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

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.

Custom cursor patterns cannot be used with GPR_$LOCATOR_UPDATE.

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_$SET_OBSCURED_OPT and specified an option of GPR_$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. See the GPR Data Types section for more information.

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

USAGE

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

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.

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

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

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.

FORMAT

```
GPR_$INIT (op_mode, unit, size, hi_plane_id, init_bitmap_desc, status)
```

INPUT PARAMETERS

op_mode

One of four modes of operation. Graphics primitives routines can operate in two borrow-display modes, within a Display Manager window, within a frame of a Display Manager pad, or without using the display. Use GPR_$DISPLAY_MODE_T format for this parameter. This is a 2-byte integer. Possible values for this parameter are:

GPR_$BORROW
: Program borrows the full screen and the keyboard from the Display Manager and uses the display driver directly through GPR software.

GPR_$BORROW_NC
: Same as GPR_$BORROW except that all the pixels are not set to zero. (screen is not cleared.)

GPR_$DIRECT
: Program borrows a window from the Display Manager instead of borrowing the whole display.

GPR_$FRAME
: Program executes within a frame of a Display Manager Pad.

GPR_$NO_DISPLAY
: GPR allocates a bitmap in main memory. No graphics is displayed on the screen.

unit

This parameter has three possible meanings, as follows:

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

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

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

size

The size of the initial bitmap (and the size of the frame, in frame mode), in GPR_$OFFSET_T format. This data type is 4 bytes long. See the GPR Data Type section for more information. Possible values are listed below.

	X	Y
Borrow-display or direct mode (limits are reduced to display or window size if necessary):	1 to 1024	1 to 1024
Display Manager Frame:	1 - 32767	1 - 32767
Main Memory Bitmap:	1 - 8192	1 - 8192

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

```
For display memory bitmaps:

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


For main memory bitmaps:

0 - 7 for all displays
```

OUTPUT PARAMETERS

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

status
Completion status, in STATUS_$T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

USAGE

To use multiple windows, you must call GPR_$INIT for each window.

GPR_$BORROW_NC allows you to allocate a bitmap in display memory without setting all the pixels to zero.

In GPR_$NO_DISPLAY mode, the program can manipulate only main memory bitmaps.

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

To use imaging formats, a program must be initialized in borrow-display 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. This data type is 4 bytes long. See the GPR Data Type section for more information.

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. See the GPR Data Types section for more information.

hi_plane_id
The identifier of the bitmap's highest plane, in GPR_$PLANE_T format. This is a 2-byte integer.

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

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

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 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 frame.

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. See the GPR Data Types section for more information.

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

USAGE

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

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

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 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.

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_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 of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

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

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_$BW_800x1024	monochromatic portrait
GPR_$BW_1024x800	monochromatic landscape
GPR_$COLOR_1024x1024x4	color 1024 x 1024 (DN6xx) 2-board config
GPR_$COLOR_1024x1024x8	color 1024 x 1024 (DN6xx) 3-board config
GPR_$COLOR_1024x800x4	color 1024 x 800 (DN5xx) 2-board config
GPR_$COLOR_1024x800x8	color 1024 x 800 (DN5xx) 3-board config
GPR_$COLOR1_1024X800X8	color 1024 x 800 (DN570) 2-board config
GPR_$COLOR_1280X1024X8	color 1280 x 1024 (DN580) 2-board config
GPR_$COLOR2_1024X800X4	color 1024 x 800 (DN3000) 1-board config

status

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

USAGE

GPR_$INQ_CONFIG can be used before GPR_$INIT. This is useful to determine the number of possible planes in bitmaps on color displays before initializing GPR.

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. See the GPR Data Type section for more information.

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

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

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

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. See the GPR Data Type section for more information.

origin
The pixel currently set as the cursor origin, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Type section for more information.

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-display mode, the x- and y-coordinates represent an absolute position on the screen. If a program calls this routine when the cursor is inside a frame of a display manager pad, the x- and y-coordinates are relative to the top left corner of the frame.

To alter the cursor, use one of the following:

```
GPR_$SET_CURSOR_PATTERN
GPR_$SET_CURSOR_ACTIVE
GPR_$SET_CURSOR_POSITION
GPR_$SET_CURSOR_ORIGIN
```

Currently, a program can not alter the cursor raster operation.

If no cursor pattern has been set, the default rectangle cursor is returned.

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,unit_or_pad,disp_len,disp,disp_len_ret,status)
```

INPUT PARAMETERS

op

One of four modes of operation. Graphics primitives routines can operate in two borrow-display modes, within a Display Manager window, within a frame of a Display Manager pad, or without using the display. Use GPR_$DISPLAY_MODE_T format for this parameter. This is a 2-byte integer. Possible values for this parameter are:

GPR_$BORROW
: Returns information about to a borrowed display.

GPR_$BORROW_NC
: Returns information about to a borrowed display.

GPR_$DIRECT Returns information about to a direct-mode window.

GPR_$FRAME Returns information about to a frame of a Display Manager Pad.

GPR_$NO_DISPLAY
: Returns infomation about to a main-memory bitmap.

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) 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 is an array of up 56 bytes. See the GPR data types section for more information.

disp_len_ret

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 28 16-bit words, or 56 bytes, so 56 is the current maximum possible value for this parameter.

status

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

USAGE

Prior to SR9.2, programs using GPR could only obtain a value that identified a particular display type, for example, a monochrome display, 1024 by 800 pixels. Programs then derived the particular display characteristics from this value. As a result, a program that wanted to determine display characteristics had to assign a value to each device type that it might want to obtain. Each time we added new display types, user programs had to be modified to identify the new display types.

GPR_$INQ_DISP_CHARACTERISTICS eliminates the need for user programs to include values that identify display device characteristics. This call returns all of a node's display characteristics as a data item in the "disp" parameter. If you use this call, you will not need to extend your programs to support any future display types.

You can call GPR_$INQ_DISP_CHARACTERISTICS at any time, regardless of whether or not GPR has been initialized. If you have initialized GPR, calling this routine has no effect on the current bitmap or its attributes.

When the program calls GPR_$INQ_DISP_CHARACTERISTICS, the values it specifies in the first two parameters are the same as the values it specifies to GPR_$INIT. These parameters identify the display mode and unit or stream to the call, which can then return specific information about the window or bitmap to be used, as well as general information about the display device. The application program must supply a buffer variable, typically of a record type in Pascal, a structure type in C, or an array type in FORTRAN, in which the data can be returned.

In the future, we may extend the list of data items that this call returns as we release new display devices with new characteristics. However, programs written to use the existing set of characteristics will continue to operate correctly.

GPR_$INQ_DRAW_VALUE

Returns the color/intensity value used for drawing lines.

FORMAT

```
GPR_$INQ_DRAW_VALUE (index, status)
```

OUTPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for drawing lines, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel Format
0 - 255	For color displays in 8-bit or 24-bit pixel Format
-1	For all displays. This specifies that the background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the line drawing value. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

status

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

USAGE

To set a new draw value, use GPR_$SET_DRAW_VALUE.

GPR_$INQ_FILL_BACKGROUND_VALUE

Returns the color/intensity value of the background used for tile fills.

FORMAT

```
GPR_$INQ_FILL_BACKGROUND_VALUE (index, status)
```

OUTPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for tile fills, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-1	For all displays. This specifies that the background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the tile fill background. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

status

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

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

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/intensity value used to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
GPR_$INQ_FILL_VALUE (index, status)
```

OUTPUT PARAMETERS

index

The color map index that indicates the current color/intensity fill value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format

status

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

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 parameter for horizontal spacing of the specified font. This is a 2-byte integer. Possible values are in the range -127 - 127.

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

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

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

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 GRP_$DOTTED.

GPR_$INQ_LINESTYLE

Returns information about the current line-style.

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

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

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. See the GPR Data Types section for more information.

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

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

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

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

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

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 chararacter 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. See the GPR Data Types section for more information.

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

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 height is the y-offset.

To change the direction of text, use GPR_$SET_TEXT_PATH.

Figure GPR-1 shows two examples of the extent of text in relation to offsets. For horizontal text, use GPR_$RIGHT with GPR_$SET_TEXT_PATH. For rotated text, use GPR_$UP with GPR_$SET_TEXT_PATH.

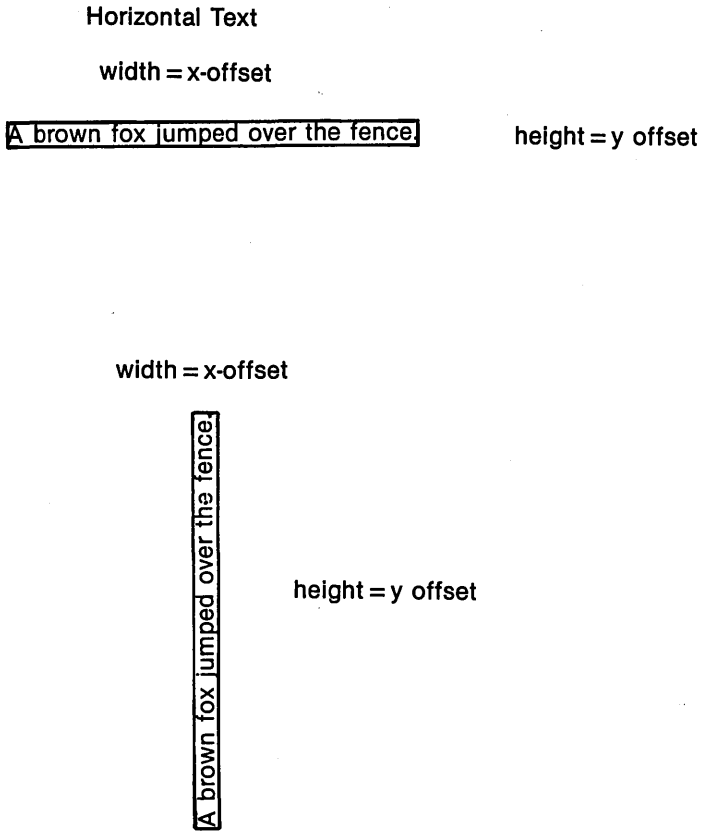


Figure GPR-1. Height and Width for Horizontal and Rotated Text

GPR_$INQ_TEXT_OFFSET

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

FORMAT

```
GPR_$INQ_TEXT_OFFSET (string, string_length, start, xy_end, status)
```

INPUT PARAMETERS

string
A string, in GPR_$STRING_T format. This is 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. See the GPR Data Type section for more information.

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

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.

Figure GPR-2 shows an example of text offsets, after using GPR_$RIGHT and GPR_$UP with GPR_$SET_TEXT_PATH.

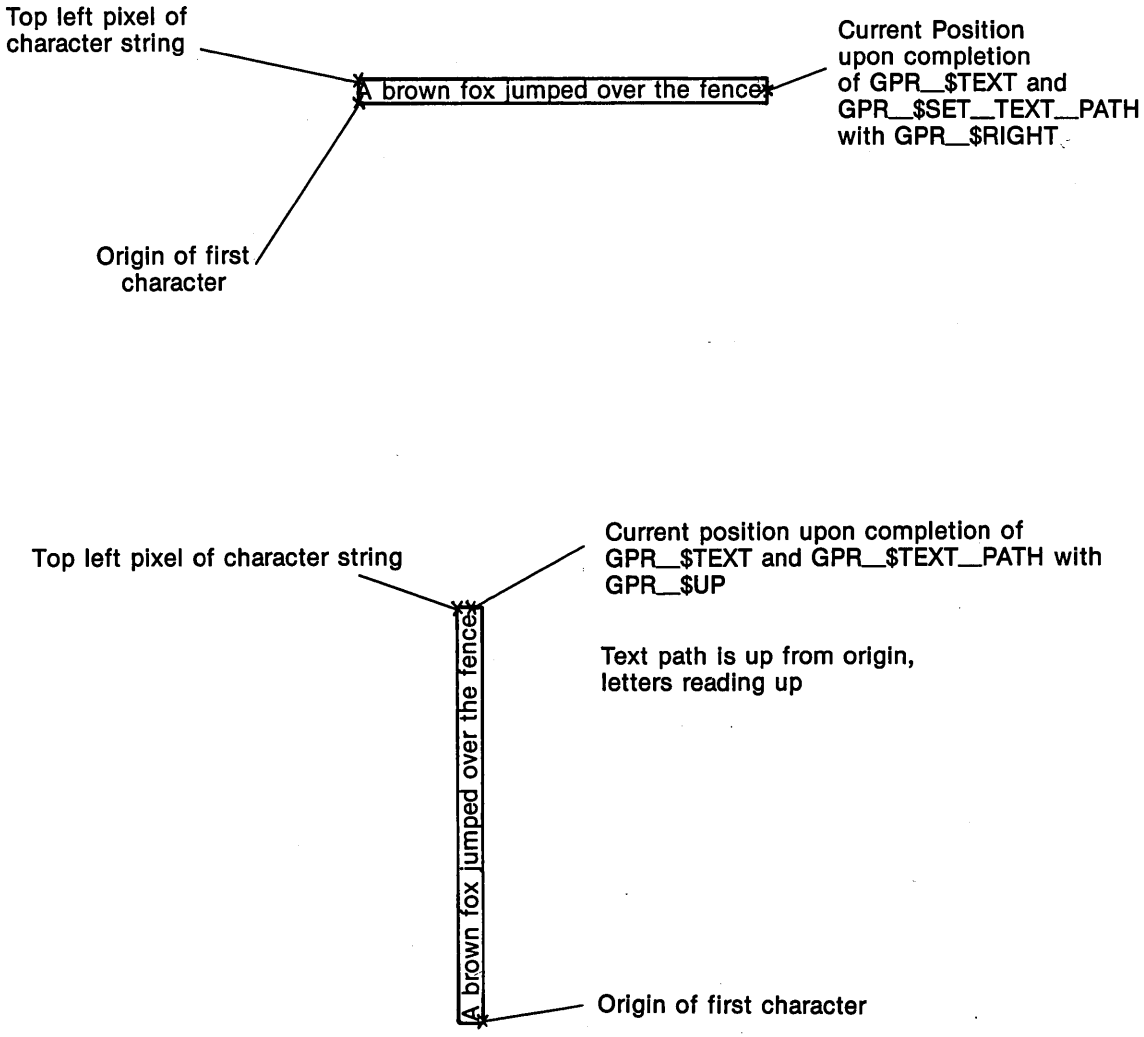


Figure GPR-2. Text Offsets

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

USAGE

To set the current text path, use GPR_$SET_TEXT_PATH.

GPR_$INQ_TEXT_VALUES

Returns the text color/intensity value and the text background color/intensity value used in the current bitmap.

FORMAT

```
GPR_$INQ_TEXT_VALUES (text_value, text_bkgd_value, status)
```

OUTPUT PARAMETERS

text_value
A color map index that indicates the text color/intensity value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer.

text_bkgd_value
A color map index that indicates the text background color/intensity value, in GPR_$PIXEL_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 the text color/intensity value, use GPR_$SET_TEXT_VALUE. To establish the text background color/intensity value, use GPR_$SET_TEXT_BACKGROUND_VALUE.

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

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.

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. See the GPR Data Types section for more information.

There is no set limit to the number of visible regions that may be returned.

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

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

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

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. Additional information on fonts can be found in the Command Reference manual.

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

USAGE

Use the font-id returned from this file as input for GPR_$SET_TEXT_FONT.

To unload fonts loaded with this routine, use GPR_$UNLOAD_FONT_FILE.

GPR_$MOVE

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

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

USAGE

GPR_$MULTILINE alternately moves to a new position and draws lines: it moves to the first given position, draws a line from the first to the second given position, updates the current position, moves to the third position, etc. After the last line has been drawn or the last move has been made, the endpoint becomes the current position.

The given coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate position is the destination of the multiline drawn.

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

GPR_$MULTITRAPEZOID

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

FORMAT

```
GPR_$MULTITRAPEZOID (trapezoid_list, trapezoid_number, status)
```

INPUT PARAMETERS

trapezoid_list
Trapezoids to fill, in GPR_$TRAP_LIST_T format. This data type is 12 bytes long. See the GPR Data Types section for more information.

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

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

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.

Filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS contain fewer pixels than filled areas rasterized with the decomposition technique set to either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS OVERLAP.

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. See the GPR Data Types section for more information.

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

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

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 equal 1. 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.

Filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris contain fewer pixels than filled areas rasterized with the decomposition technique set to either gpr_$fast_traps or gpr_$precise_traps.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either gpr_$fast_traps or gpr_$precise_traps overlap.

GPR_$OPEN_BITMAP_FILE

Opens a file for external storage of a bitmap.

FORMAT

```
GPR_$OPEN_BITMAP_FILE (access, filename, name_size, version, size, groups,
                       group_header, attributes, 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 only 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.

name_size
The length of the file name. This is a 2-byte integer.

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.

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

groups
The number of groups in external bitmaps. This is a 2-byte integer. Possible values are 1..(GPR_$MAX_BMF_GROUP +1). Currently, a bitmap can contain only 1 group.

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.

N_SECTS — The number of sections in the group. Currently, there must be 1 section for each plane of a bitmap. N_SECTS is a 2-byte integer which can have a value in the range 1 - 8.

PIXEL_SIZE — The number of bits per pixel in each section of a group. Since each section currently can contain only 1 plane of a bitmap, this value must be 1. PIXEL_SIZE is a 2-byte integer.

ALLOCATED_SIZE
2-byte integer Currently, this value must be 1, but you can specify this value as 0 and GPR will perform the necessary calculations.

BYTES_PER_LINE
The number of bytes in one row of one plane of the bitmap. BYTES_PER_LINE is a 2-byte integer. The value must be a multiple of 4, but can be specified as 0 and GPR will perform the necessary calculations.

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. This value can be specified as 0 and GPR will perform the necessary calculations.

STORAGE_OFFSET
UNIV_PTR format

INPUT PARAMETERS

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

USAGE

Currently, a section is equivalent to one plane of a bitmap. N_SECTS may include up to eight bit planes.

For ALLOCATED_SIZE, BYTES_PER_LINE and BYTES_PER_SECT, you can specify values as 0, and the GPR package will calculate and return the appropriate values.

BYTES_PER_SECT is not necessarily a multiple of BYTES_PER_LINE. This means that GPR will leave unused space at the end of one section to satisfy alignment constraints. The result is that the next section starts on an alignment boundary, which is normally a page boundary.

The access parameter specifies one of four ways to use external bitmaps. As shown in the table below, the value given for this parameter determines whether four other parameters are input (IN) or output (OUT). The values for these parameters are used to validate your input with GPR_$CREATE and GPR_$UPDATE.

	GPR_$CREATE	GPR_$UPDATE		GPR_$WRITE	GPR_$READONLY
		file exists			
		no	yes		
version, size, groups, group-headers	IN	IN	OUT	OUT	OUT

GPR_$CREATE indicates that you want a new external bitmap file. GPR_$UPDATE means that you want to create a new file or overwrite an existing one.

When you specify GPR_CREATE as the access parameter and you specify a file name that already exists, the file is superseded only if it is a bitmap file. If the file is not a bitmap file, you get the error message NAME_$ALREADY_EXISTS.

Attributes are not stored with the bitmap. You assign attributes when you open the bitmap file. See the routines GPR_$ALLOCATE_ATTRIBUTE_BLOCK and GPR_$ALLOCATE_BITMAP.

Figure GPR-3 is a global view of one group.

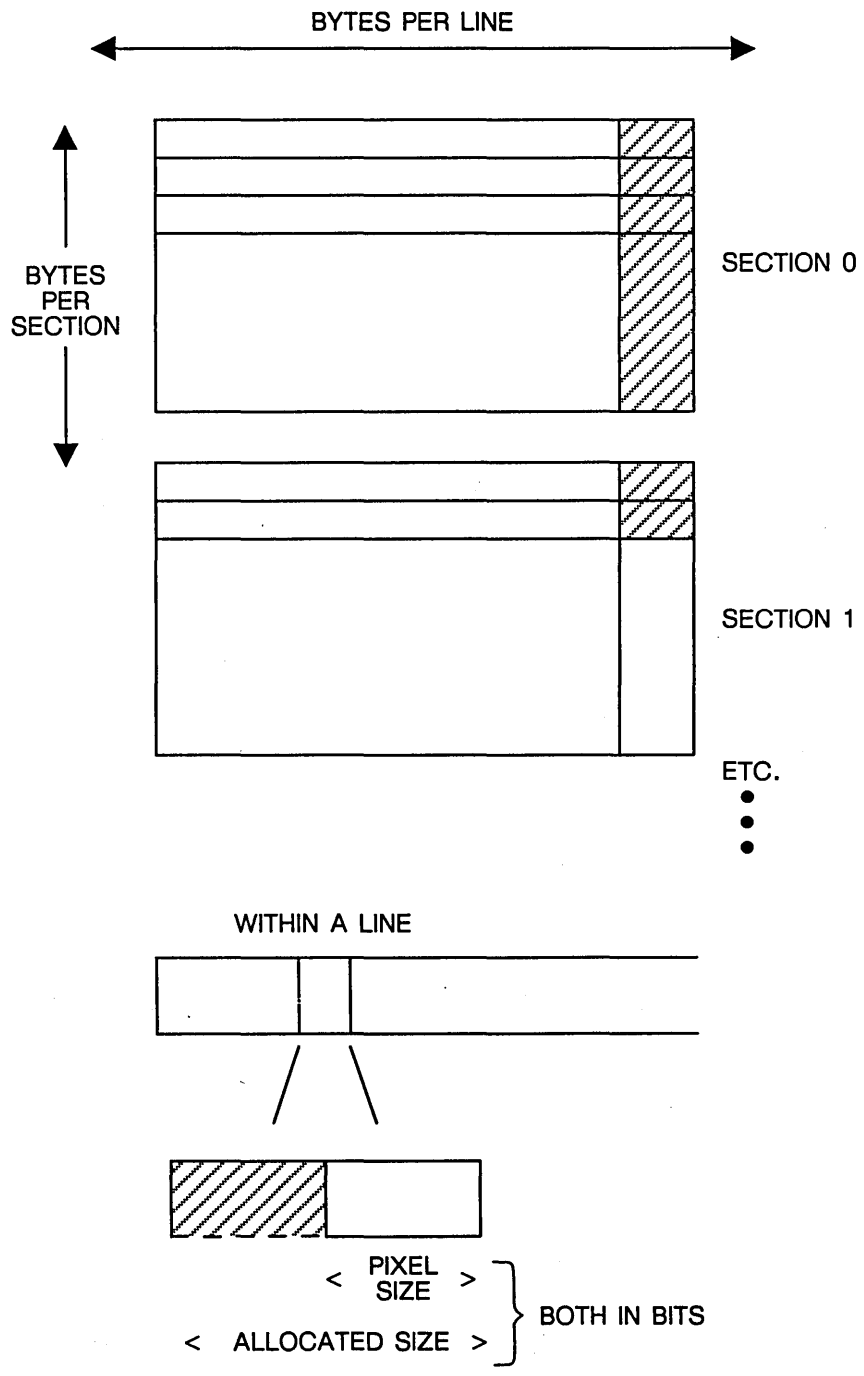


Figure GPR-3. View of One Group

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

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 Appendix E of Programming with DOMAIN Graphics Primitives for infomation 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.
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. This data type is 4 bytes long. See the GPR Data Types section for more information.

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. See the GPR Data Types section for more information.

dest_origin
Start position (top left coordinate position) of the destination rectangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

Use GPR_$SET_BITMAP to establish the current bitmap for this routine.

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

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

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

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. See the GPR Data Types section for more information.

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

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. See the GPR Data Types section for more information.

OUTPUT PARAMETERS

pixel_array
An array of the pixel values (color/intensity) in GPR_$PIXEL_ARRAY_T format. This is a 131,073-element array of 4-byte integers.

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

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 pixels 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. See the GPR Data Type section for more information.

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

GPR_$RECTANGLE fills in a rectangle with the color/intensity value specified with GPR_$SET_FILL_VALUE. To retrieve the current fill value, use GPR_$INQ_FILL_VALUE.

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

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_$PLANE_T. This is a 2-byte integer. A pointer can be returned to the plane using GPR_$INQ_BITMAP_POINTER. Valid values are 0 - 7.

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

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 GPR_INQ_BITMAP_POINTER is used. 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_$PLANE_T. This is a 2-byte integer. A pointer can be returned to the plane using GPR_$INQ_BITMAP_POINTER. Valid values are 0 - 7.

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

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/580 and DN3000.

GPR_$REPLICATE_FONT

Creates and loads a modifiable copy of a font.

FORMAT

```
GPR_$REPLICATE_FONT (font_id, repli_font_id, status)
```

INPUT PARAMETERS

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

OUTPUT PARAMETERS

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

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

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

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_$SET_ACQ_TIME_OUT

Establishes the length of time the display will be acquired.

FORMAT

```
GPR_$SET_ACQ_TIME_OUT (timeout, status)
```

INPUT PARAMETERS

timeout
The maximum real time, in TIME_$CLOCK_T format, for which the program can acquire the display.

OUTPUT PARAMETERS

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

USAGE

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

If this routine is not called, the default time-out value is one minute.

GPR_$SET_ATTRIBUTE_BLOCK

Associates an attribute block with the current bitmap.

FORMAT

```
GPR_$SET_ATTRIBUTE_BLOCK (attrib_block_desc, status)
```

INPUT PARAMETERS

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

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

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

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. See the GPR Data Types section for more information.

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

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
Monochromatic display (either portrait or landscape)	1024	1024	0
Color display--Interactive format			
4-bit pixels	1024	2048	3
8-bit pixels	1024	2048	7

If a program uses hidden display memory, it must be careful not to modify areas that are being used to store fill constants or text fonts. The following areas may be used by these functions on the various DOMAIN displays.

Fill constants:

Both monochromatic displays: 800 <= X <= 1023 and Y = 1023.

Color displays: none.

Stand-alone font:

Monochromatic portrait display: 800 <= X <= 1023 and 0 <= Y <= 39.

Monochromatic landscape display: 800 <= X <= 1023 and 983 <= Y <= 1022.

Color displays: same as monochromatic portrait display, plane 0 only, Y offset by 1024.

User text fonts: (only if text fonts are loaded)

Monochromatic portrait display: 800 <= X <= 1023 and 40 <= Y <= 1022, allocated from top to bottom.

Monochromatic landscape display: 0 <= X <= 1023 and 800 <= Y <= 1023, in columns 224 bits wide, allocated top to bottom and left to right.

Color displays: same as monochromatic portrait display, plane 0 only, Y offset by 1024.

Note that these areas may move, grow or shrink in future DOMAIN software releases. Therefore, only limited use should be made of hidden display memory in conjunction with text or cursor operations.

GPR_$SET_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 long integers (4-byte integers).

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

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 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 parameter of the specified character. This is a 2-byte integer. Possible values are -127 to 127.

OUTPUT PARAMETERS

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

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. See GPR Data Types section for more information.

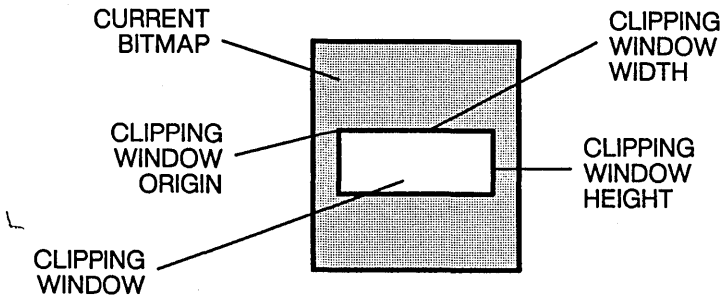


Figure GPR-4 Clipping Window Origin, Width, Height

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

The default clip window is the entire bitmap.

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 enable the clip window, use GPR_$SET_CLIPPING_ACTIVE.

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

USAGE

To specify a clipping window, use the routine GPR_$SET_CLIP_WINDOW.

Initially, in borrow-display, 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-bit pixel format
1 - 256	For color displays in 8-bit or 24-bit pixel format

values
Color value entries, in GPR_$COLOR_VECTOR_T format. This is a 256-element array of 4-byte integers.

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

For the monochromatic display, the default start-index is 0, n-entries is 2, and the values are GPR_$BLACK and GPR_$WHITE. Dark has the value GPR_$BLACK, and bright has the value GPR_$WHITE. A program can use this routine to redefine the pixel values corresponding to bright and dark intensity.

For the monochromatic display, if the program provides fewer than two values, or if the first two values are the same (both black or both white), the routine returns an error.

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.

On a monochrome system that uses a simulated color map 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 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.

In direct mode, you must acquire the display before establishing new values for the color map.

To retrieve the current color map, use GPR_$INQ_COLOR_MAP.

GPR_$SET_COORDINATE_ORIGIN

Establishes x- and y-offsets to add to all x- and y-coordinates used 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. See the GPR Data Types section for more information.

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 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.

This routine may not be used on a bitmap corresponding to a Display Manager frame.

GPR_$SET_CURSOR_ACTIVE

Specifies whether the cursor is displayed.

FORMAT

```
GPR_$SET_CURSOR_ACTIVE (active, status)
```

INPUT PARAMETERS

active
Boolean (logical) value that specifies whether to display the cursor. Set the parameter to true to display the cursor; set it to false if you do not want to display the cursor.

OUTPUT PARAMETERS

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

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-display or direct mode.

GPR_$SET_CURSOR_ORIGIN

Defines one of the cursor's pixels as the cursor origin.

FORMAT

```
GPR_$SET_CURSOR_ORIGIN (origin, status)
```

INPUT PARAMETERS

origin
The position of one cursor pixel (the origin) relative to the entire cursor, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

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

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. See the GPR Data Types section for more information.

The first element is the cursor position's x-coordinate; the second element is the y-coordinate. Coordinate values must be within the limits of the display in use, as follows:

	X	Y
Borrowed Display:		
Monochromatic Portrait:	0 - 799	0 - 1023
Monochromatic Landscape:	0 - 1023	0 - 799
Color:	0 - 1023	0 - 1023
Color : 550	0 - 1023	0 - 799
Display Manager Frame:	0 - 32767	0 - 32767

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-display mode, the x- and y-coordinates represent an absolute position on the screen. If a program calls this routine when the cursor is inside a frame of a Display Manager pad, the x- and y-coordinates are offsets from the top left corner of the frame.

If the coordinate position would cause any part of the cursor to be outside the screen or frame, the cursor moves only as far as the edge of the screen. The cursor is neither clipped nor made to disappear.

To request the current cursor position, use GPR_$INQ_CURSOR.

In a Display Manager frame, this routine moves the cursor only if the cursor is in the window viewing this frame when the call is issued. If not, a "next window" command which moves to that window will move the cursor to its new position.

GPR_$SET_DRAW_VALUE

Specifies the color/intensity value to use to draw lines.

FORMAT

```
GPR_$SET_DRAW_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for drawing lines, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the line drawing value. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status

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

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_FILL_BACKGROUND_VALUE

Specifies the color/intensity value used for drawing the background of tile fills.

FORMAT

```
GPR_$SET_FILL_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for tile fills, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-1	For all displays. This specifies that the fill background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the fill background. For borrowed displays and memory bitmaps, the fill background is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status

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

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/intensity value to use to fill circles, rectangles, triangles, and trapezoids.

FORMAT

```
GPR_$SET_FILL_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current fill color/intensity value, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. The default fill value is 1. Valid values are:

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

OUTPUT PARAMETERS

status

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

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.

"Index" is a color map index, not a color value.

GPR_$SET_HORIZONTAL_SPACING

Specifies the parameter for horizontal spacing of the specified font.

FORMAT

```
GPR_$SET_HORIZONTAL_SPACING (font_id, horizontal_spacing, status)
```

INPUT PARAMETERS

font_id
The identifier of the text font. This is a 2-byte integer.

horizontal_spacing
The horizontal spacing parameter of the specified font. This is a 2-byte integer. Possible values are -127 - 127.

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

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

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.

GPR_$SET_IMAGING_FORMAT accepts only GPR_$INTERACTIVE on the following models: DN570/570A/580 and DN3000.

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

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

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

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

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

USAGE

The default mask specifies that all planes are used.

Operations occur only on the planes specified in the mask. A program can use this routine, for example, to perform raster operations on separate planes or groups of planes in the bitmap.

Using the mask, a program can partition the 8-bit pixels into subunits. For example, the program can use planes 0 - 3 for one picture and planes 4 - 7 for another. Thus, one bitmap may contain two color pictures. This does not, however, increase the number of colors available for one bitmap.

To retrieve the current plane mask, use GPR_$INQ_CONSTRAINTS.

GPR_$SET_RASTER_OP

Specifies a 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. See GPR Data Types section for more information.

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

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

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 effected 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.

The *Programming With General System Calls* describes the pointer data type.

See *Programming With DOMAIN Graphic Primitives* for an algorithm using procedure pointers.

GPR_$SET_SPACE_SIZE

Specifies the size of horizontal spacing for the specified font.

FORMAT

```
GPR_$SET_SPACE_SIZE (font_id, space_size, status)
```

INPUT PARAMETERS

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

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

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 which change fonts, you must first call GPR_$REPLICATE_FONT to create a modifiable copy of a font. The font-modifying routines include GPR_$SET_CHARACTER_WIDTH, GPR_$SET_HORIZONTAL_SPACING, and GPR_$SET_SPACE_SIZE. These calls change only the local copy of the font. If you unload a font and reload it, the font is reset to the values in the font file.

The space size is the number of pixels to skip in the horizontal direction when you 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/intensity value to use for text background.

FORMAT

```
GPR_$SET_TEXT_BACKGROUND_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for the text background, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. This parameter is an index into a color map; it is not a color value. Valid values are:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format
-1	For all displays. This specifies that the text background is transparent; that is, the old values of the pixels are not changed.
-2	For all displays. This specifies using the color/intensity value of the bitmap background as the text background. For borrowed displays and memory bitmaps, this value is always zero. For Display Manager frames, this is the pixel value in use for the window background.

OUTPUT PARAMETERS

status

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

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

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

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/intensity value to use for writing text.

FORMAT

```
GPR_$SET_TEXT_VALUE (index, status)
```

INPUT PARAMETERS

index

The color map index that indicates the current color/intensity value used for writing text, in GPR_$PIXEL_VALUE_T format. This is a 4-byte integer. The valid values are listed below:

0 - 1	For monochromatic displays
0 - 15	For color displays in 4-bit pixel format
0 - 255	For color displays in 8-bit or 24-bit pixel format

OUTPUT PARAMETERS

status

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

USAGE

To retrieve the current text value, use GPR_$INQ_VALUES.

The default text value is 1 for borrowed displays, memory bitmaps, and Display Manager frames on monochromatic displays; 0 for Display Manager frames on color displays.

For monochromatic displays, only the low-order bit of the text value is considered, because monochromatic displays have only one plane.

For color displays in 4-bit pixel format, only the four lowest-order bits of the text value are considered, because these displays have four planes.

The color specification parameter is a color map index, not a color value.

GPR_$SET_TRIANGLE_FILL_CRITERIA

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

FORMAT

```
GPR_$SET_TRIANGLE_FILL_CRITERIA(fill_crit, status)
```

INPUT PARAMETERS

fill_crit

Sets 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 non-zero 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.

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

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.

GPR_$SET_WINDOW_ID

Establishes the character that identifies the current bitmap's window.

FORMAT

```
GPR_$SET_WINDOW_ID (character, status)
```

INPUT PARAMETERS

character
The character that identifies the current bitmaps's window. This is a character variable.

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

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

USAGE

GPR_$SPLINE_CUBIC_P draws a smooth curve starting from the current position, through each of the specified points.

After the spline is drawn, the last point becomes the current position.

The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

An error is returned if any two consecutive points are equal.

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

GPR_$SPLINE_CUBIC_X

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

FORMAT

```
GPR_$SPLINE_CUBIC_X (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions.
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. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$SPLINE_CUBIC_X draws a smooth curve starting from the current position and through each of the specified points.

After the spline is drawn, the last point becomes the current position.

The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

An error is returned if any x-coordinate is less than or equal to a previous x-coordinate. The x-coordinate array must be sorted into increasing order.

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

GPR_$SPLINE_CUBIC_Y

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

FORMAT

```
GPR_$SPLINE_CUBIC_Y (x, y, npositions, status)
```

INPUT PARAMETERS

x

List of the x-coordinates of all the successive positions.
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. This data type is 4 bytes long. See the GPR Data Types section for more information.

USAGE

GPR_$SPLINE_CUBIC_Y draws a smooth curve starting from the current position and through each of the specified points.

After the spline is drawn, the last point becomes the current position.

The specified coordinates are added to the corresponding elements of the coordinate origin for the current bitmap. The resultant coordinate positions are the points through which the spline is drawn.

An error is returned if any y-coordinate is less than or equal to a previous y-coordinate. The y-coordinate array must be sorted into increasing order.

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

GPR_$START_PGON

Defines the starting position of a polygon.

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

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 using either GPR_$CLOSE_FILL_PGON or GPR_$CLOSE_RETURN_PGON and GPR_$MULTITRAPEZOID.

This routine closes any previously open loop of edges by connecting its last endpoint to its first endpoint with an edge. Then, the routine starts the new loop.

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

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

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 a next character would be written. This is the case even if the string is partly or completely clipped. However, the current position always remains within the boundaries of the bitmap.

GPR_$TRAPEZOID

Draws and fills a trapezoid.

FORMAT

```
GPR_$TRAPEZOID (trapezoid, status)
```

INPUT PARAMETERS

trapezoid
Trapezoid in GPR_$TRAP_T format. This data type is 12 bytes long. See the GPR Data Types section for more information.

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

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.

Filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS contain fewer pixels than filled areas rasterized with the decomposition technique set to either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS.

Abutting filled areas rasterized when the decomposition technique is GPR_$NON_OVERLAPPING_TRIS do not overlap.

Abutting filled areas rasterized when the decomposition technique is either GPR_$FAST_TRAPS or GPR_$PRECISE_TRAPS OVERLAP.

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

vertex_2
Second vertex of the triangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

vertex_3
Third vertex of the triangle, in GPR_$POSITION_T format. This data type is 4 bytes long. See the GPR Data Types section for more information.

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

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.

Filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris contain fewer pixels than filled areas rasterized with the decomposition technique set to either gpr_$fast_traps or gpr_$precise_traps.

Abutting filled areas rasterized when the decomposition technique is gpr_$non_overlapping_tris do not overlap.

Abutting filled areas rasterized when the decomposition technique is either gpr_$fast_traps or gpr_$precise_traps overlap.

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

USAGE

The font_id is returned when a program loads a file with the routine GPR_$LOAD_FONT_FILE.

GPR_$WAIT_FRAME

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

FORMAT

```
GPR_$WAIT_FRAME (status)
```

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

This routine is for use on color displays only.

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
A 131,073-element array of 4-byte integers in GPR_$PIXEL_ARRAY_T format from which to write pixel values (color/intensity).

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. See the GPR Data Types section for more information.

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

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
: Primitives are already initialized.

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.

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.

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

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_$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_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_$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

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 Primitive Resource Call Reference*
Order No.: 007194 Revision: 01 Date of Publication: January, 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

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 Primitive Resource Call Reference*
Order No.: 007194 Revision: 01 Date of Publication: January, 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

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 Primitive Resource Call Reference*
Order No.: 007194 Revision: 01 Date of Publication: January, 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