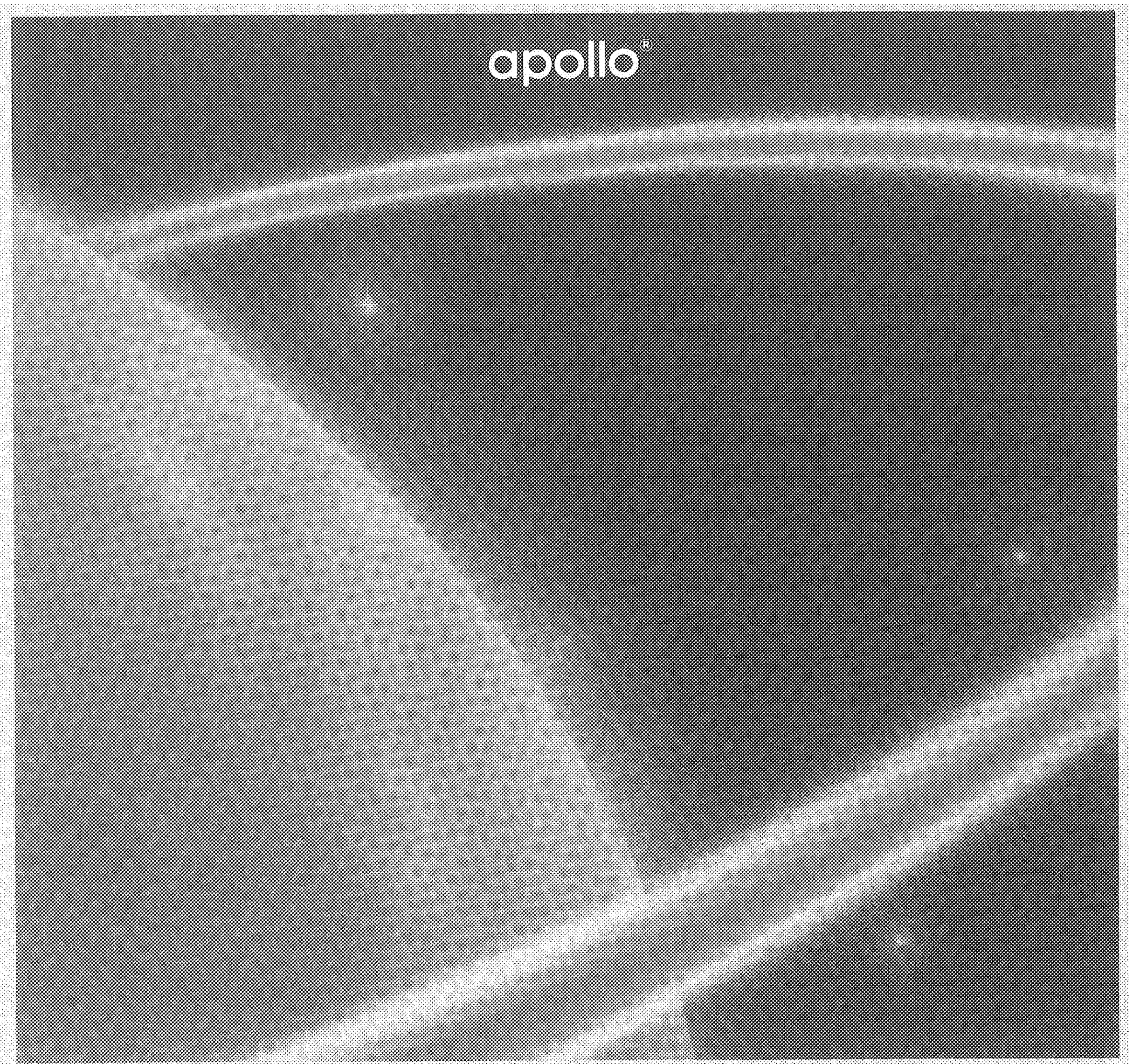
apollo®



D O M A I N

Order No. 009793
Revision 00

DOMAIN 2D Graphics Metafile Resource Call Reference

DOMAIN 2D Graphics Metafile Resource Call Reference

Order No. 009793
Revision 00

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

First Printing: November, 1986

Preface

The *DOMAIN 2D Graphics Metafile Resource Call Reference* 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.

Audience

This manual is for programmers who use the DOMAIN 2D Graphics Metafile Resource to develop application programs. Users of this manual 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 2D Graphics Metafile Resource* before using this reference.

Organization of this Manual

This manual contains four chapters:

Chapter 1	Presents the constants and data types used by the 2D Graphics Metafile Resource package.
Chapter 2	Presents a description of each routine including format and parameters. The organization of routines is alphabetical.
Chapter 3	Presents a listing of 2D GMR errors and a brief description of each error.
Chapter 4	Presents two listings of 2D GMR routines. The first is a listing of routines and descriptions by function. The second is an alphabetical listing of call formats.

Additional Reading

Use this reference as a companion to the *Programming With 2D Graphics Metafile Reference* manual (005097).

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

The *Programmer's Guide to DOMAIN Graphics Primitives* manual (005808) 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.

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).

The 2D GMR package creates POSTSCRIPT files for hardcopy output to laser printers that support POSTCRIPT. If you want to modify the POSTSCRIPT files, see the *POSTSCRIPT Language Reference* (007765).

You can use 2D GMR with the DOMAIN/Dialogue user interface. See the *DOMAIN/Dialogue User's Guide* (004299).

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.

Contents

Chapter 1
Constants and Data Types

This chapter describes the constants and data types used by the 2D Graphics Metafile Resource package (hereafter referred to as 2D GMR). Each data type description includes an atomic data type translation (i.e., GM_$ACC_CREATE_T = 2-byte integer) as well as a brief description of the type's purpose. The description includes any predefined values associated with the type. The following is an example of a data type description for the GM_$CONC_MODE_T type:

GM_$CONC_MODE_T

A 2-byte integer. Defines the number of concurrent users a file may have. One of the following predefined values:

GM_$1W
N readers or 1 writer is allowed.

GM_$COWRITERS
More than 1 writer is allowed,but all must be on the same node.

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

GM_$MAX_ACLASS	16	The maximum number of attribute classes is 16.
GM_$MAX_ARRAY_LENGTH	1000	The maximum number of elements in a gm_$array..._t is 1000.
GM_$MAX_BLOCK	40	The maximum number of attribute blocks is 40.
GM_$MAX_CURSOR_PATTERN_WORDS	16	The maximum number of words in a cursor pattern.
GM_$MAX_DRAW_PATTERN_BYTES	8	The maximum number of bytes in a draw pattern.
GM_$MAX_FILE	16	The maximum number of files is 16.
GM_$MAX_FILL_PATTERN_LWORDS pattern.	32	The max number of long words in a fill
GM_$MAX_FONT	32	The maximum number of font family identification numbers is 32.
GM_$MAX_FONT_FAMILY	8	The maximum number of font families is 8.
GM_$MAX_GRID	4	The maximum number of grids that may be associated with a vewport.
GM_$MAX_INSTANCE_DEPTH	32	The maximum instancing depth.
GM_$MAX_PIXEL_VALUE	255	The maximum value for color map entries; the numbers are 0 through 255.
GM_$MAX_PLANE_ID	7	The maximum number of planes.
GM_$MAX_PRIM_ID	16	The maximum number of primitive commands is 16.
GM_$MAX_SEGMENT	65536	The maximum number of segments; the numbers are 0 through 65536
GM_$MAX_SEGMENT_ID	16#7FFFFFFF	The largest possible segment id.
GM_$MAX_SEGMENT_NAME_LENGTH	12	Maximum length for segment names is 12.
GM_$MAX_STRING_LENGTH	12	The maximum length of a GM string is 12.
GM_$MAX_VIEWPORT	64	The maximum number of viewports is 64.
GM_$OUT1_CIRCLE	16#40	Opcode to format vector output.

GM_$OUT1_CIRCLE_FILL	16#41	Opcode to format vector output.
GM_$OUT1_CURVE	16#50	Opcode to format vector output.
GM_$OUT1_DRAW_RASTER_OP	16#82	Opcode to format vector output.
GM_$OUT1_DRAW_STYLE	16#81	Opcode to format vector output.
GM_$OUT1_DRAW_VALUE	16#80	Opcode to format vector output.
GM_$OUT1_EOF	16#00	Opcode to format vector output.
GM_$OUT1_FILL_BACKGROUND_VALUE	16#91	Opcode to format vector output.
GM_$OUT1_FILL_PATTERN	16#92	Opcode to format vector output.
GM_$OUT1_FILL_VALUE	16#90	Opcode to format vector output.
GM_$OUT1_FONT_FAMILY	16#A3	Opcode to format vector output.
GM_$OUT1_PLANE_MASK	16#82	Opcode to format vector output.
GM_$OUT1_POLYLINE_2D	16#20	Opcode to format vector output.
GM_$OUT1_POLYLINE_CLOSE_2D	16#21	Opcode to format vector output.
GM_$OUT1_POLYLINE_FILL_2D	16#22	Opcode to format vector output.
GM_$OUT1_PRIMITIVE	16#60	Opcode to format vector output.
GM_$OUT1_RECTANGLE	16#30	Opcode to format vector output.
GM_$OUT1_RECTANGLE_FILL_2D	16#31	Opcode to format vector output.
GM_$OUT1_TEXT	16#70	Opcode to format vector output.
GM_$OUT1_TEXT_BACKGROUND_VALUE	16#A1	Opcode to format vector output.
GM_$OUT1_TEXT_SIZE	16#A2	Opcode to format vector output.
GM_$OUT1_TEXT_VALUE	16#A0	Opcode to format vector output.

DATA TYPES

GM_$ACC_CREATE_T

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

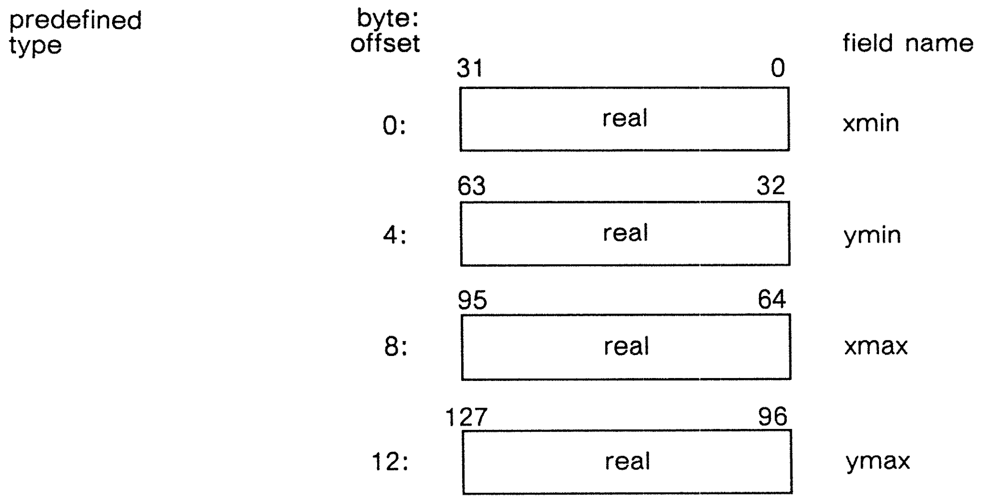
GM_$WRITE
An error is returned if the file already exists.

GM_$OVERWRITE
The previous version is deleted if the file already exists.

	GM_$UPDATE The previous version is opened if the file already exists.
GM_$ACC_OPEN_T	A 2-byte integer. Specifies the read/write accessibility. One of the following predefined values: GM_$WR Access is read or write. GM_$R Access is read only. GM_$CWR Access is read or write; if the file does not exist, create it.
GM_$ARRAY16_T	An array of 2-byte integers with MAX_ARRAY_LENGTH elements. A list of coordinate points.
GM_$ARRAY32_T	An array of 4-byte integers with MAX_ARRAY_LENGTH elements. A list of coordinate points.
GM_$ARRAYREAL_T	An array of floating-point numbers with MAX_ARRAY_LENGTH elements. A list of coordinate points.
GM_$BORDER_UNIT_T	A 2-byte integer. The units for border size. One of the following predefined values: GM_$PIXELS Expresses edge width in pixels. GM_$FRACTIONS Expresses edge width as fractions of the total GM bitmap size.

GM_$BOUNDSREAL_T

Defines the bounds of a rectangular area. The diagram below illustrates the gm_$boundsreal_t data type:



Field Description:

xmin
The x-coordinate of the bottom-left corner of the rectangle.

ymin
The y-coordinate of the bottom-left corner of the rectangle.

xmax
The x-coordinate of the top-right corner of the rectangle.

ymax
The y-coordinate of the top-right corner of the rectangle.

GM_$COLOR_ENTRY_T	A 3-element array of real values. Specifies color values in this order: red, green, blue.
GM_$COLOR_VECTOR_T	An array of 4-byte integers, of up to 256 elements. Specifies a list of color values.
GM_$COMMAND_TYPE_T	A 2-byte integer. Specifies the command type as follows: One of the following predefined values:

GM_$TACLASS
Attribute class.

GM_$TCIRCLE_2D
Circle.

GM_$TCURVE_2D
Curve.

GM_$TDRAW_RASTER_OP
Raster operations used in drawing.

GM_$TDRAWSTYLE
Line style used in drawing.

GM_$TDRAWVALUE
Pixel value used in drawing.

GM_$TFILLVALUE
Fill value used in drawing.

GM_$TFILLPATTERN
Fill pattern used in drawing.

GM_$TFONTFAMILY
Font family.

GM_$TINSTANCE_SCALE_2D
Scale and translate a segment instance.

GM_$TINSTANCE_TRANS_2D
Translate a segment instance.

GM_$TINSTANCE_TRANSFORM_2D
Rotate and translate a segment instance.

GM_$TPLANEMASK
Segment: change plane mask.

GM_$TPOLYLINE
Draw a linked set of line segments.

GM_$TPRIMITIVE
Draw a primitve.

GM_$TRECTANGLE
Draw a rectangle.

	GM_$TTAG Insert a tag. GM_$TTEXT_2D Write a text string. GM_$TTEXTBVALUE Background value for text. GM_$TTEXTSIZE Size for text. GM_$TTEXTVALUE The pixel value for text.
GM_$CONC_MODE_T	A 2-byte integer. Defines the number of concurrent users a file may have. One of the following predefined values: GM_$1W N readers or 1 writer is allowed. GM_$COWRITERS More than 1 writer is allowed, but all must be on the same node.
GM_$CURSOR_PATTERN_T	A gm_$max_cursor_pattern_words-element array of 2-byte integers. Specifies the values that set the cursor pattern.
GM_$CURSOR_STYLE_T	A 2-byte integer. Specifies the type of cursor. One of the following predefined values: GM_$BITMAP Only value is bitmap.
GM_$CURVE_T	A 2-byte integer. Specifies the type of curve. One of the following predefined values: GM_$SPLINE_CUBIC_P Draw a smooth curve through n points. GM_$ARC_3P Draw an arc through three points.
GM_$DATA_TYPE_T	A 2-byte integer. Specifies the form in which to store data. One of the following predefined values: GM_$16 Data is stored as 2-byte integers. GM_$32 Data is stored as 4-byte integers. GM_$REAL Data is stored as 4-byte integers.

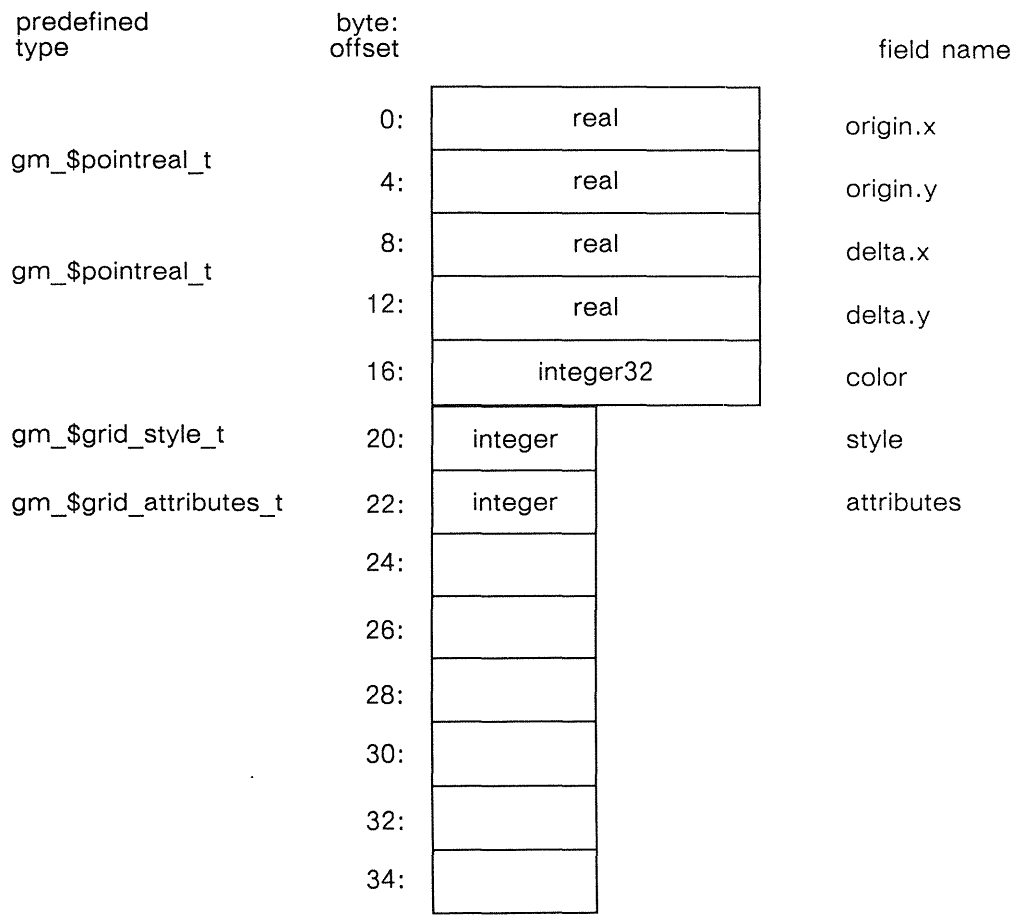
GM_$DISPLAY_CONFIG_T	A 2-byte integer. Returns the current display configuration. One of the following predefined values: GM_$BW_800X1024 A portrait black and white display. GM_$BW_1024X800 A landscape black and white display. GM_$BW_1280X1024 A landscape black and white display. GM_$COLOR_1024X1024X4 A four-plane color display. GM_$COLOR_1024X1024X8 An eight-plane color display. GM_$COLOR_1024X800X4 A four-plane color display. GM_$COLOR_1024X800X8 An eight-plane color display. GM_$COLOR_1280X1024X8 An eight-plane color display. GM_$COLOR1_1024X800X8 An eight-plane color display. GM_$COLOR2_1024X800X4 A four-plane color display. GM_$BW_1280X1024 A landscape black and white display.
GM_$DISPLAY_MODE_T	A 2-byte integer. Specifies the mode of operation. One of the following predefined values: GM_$BORROW Uses the entire screen. GM_$MAIN_BITMAP Displays within a bitmap allocated in main memory. GM_$DIRECT Displays within a Display Manager window. GM_$NO_BITMAP Allows editing of files without display. GM_$WITHIN_GPR Displays the output of the metafile within a

	bitmap that you initialize using routines of DOMAIN graphics primitives. GM_$CURRENT_BITMAP Use the current DOMAIN/Dialogue GPR bitmap.
GM_$DRAW_PATTERN_T	An array of up to gm_$max_draw_pattern_bytes characters. Specifies the bit pattern to use in drawing lines.
GM_$EVENT_T	A 2-byte integer. Specifies the type of input event; same as gpr_$event_t. One of the following predefined values: GM_$KEYSTROKE Returned when you type a keyboard character. GM_$BUTTONS Returned when you press a button on the mouse or bitpad puck. GM_$LOCATOR Returned when you move the mouse or bitpad puck or use the touchpad. GM_$ENTERED_WINDOW Returned when the cursor enters a window in which the GM bitmap resides. Direct mode only. GM_$LEFT_WINDOW Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode only. GM_$LOCATOR_STOP Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.
GM_$FILL_PATTERN_T	A gm_$max_fill_pattern_lwords-element array of 4-byte integers. Specifies the pattern to use in filling areas.
GM_$FONT_TYPE_T	A 2-byte integer. Specifies the type of font. One of the following predefined values: GM_$PIXEL A font defined by pixels. GM_$STROKE A font defined by strokes.
GM_$GRID_ARRAY_T	A list of grid specifications. See gm_$grid_t for a

	complete description and a diagram of one element of the array. This array holds up to gm_$max_grid elements.
GM_$GRID_ATTRIBUTES_T	Specifies whether the grid is placed above or below the the displayed segment. Currently only one value. GM_$GRID_ABOVE Specifies that the grid is placed above the displayed segment.
GM_$GRID_FLAGS_T	A 2-byte integer. Specifies whether the snap grid is visible/invisible and/or whether snapping is enabled. Snapping is not currently implemented. Any combination of the following predefined values: GM_$GRID_VISIBLE Indicates that the snap grid is visible. GM_$GRID_SNAPTO Indicates that the grid uses snapping.
GM_$GRID_STYLE_T	A 2-byte integer. Specifies the type of grid style to be displayed in a viewport. One of the following predefined values: GM_$GRID_POINT The grid intersections are shown by points. GM_$GRID_CROSS The grid intersections are shown by cross hairs. GM_$GRID_BOX The grid is shown as boxes GM_$GRID_AXIS The grid is orthogonal coordinate axes.

GM_$GRID_T

Specifies the characteristics of the grid. The diagram below illustrates the gm_$grid_t data type:



Field Description:

origin
The x and y coordinates of the origin of the grid in viewport segment coordinates.

delta
The delta-x and delta-y of the grid in viewport segment coordinates.

color
The color of the grid.

style
The style of the grid display. This is the tag field of the variant record. The value of style determines which fields require information if

you are establishing a grid, or which fields contain information if you are inquiring about a grid.

attributes
The attributes of the grid. Currently, the only attribute is for grid placement. The grid can be placed above or below the viewport.

If the value of style is gm_$grid_point, do not add or try to retrieve any more information. The variant part of the record contains no valid information.

If the value of style is gm_$grid_cross, insert or retrieve the cross width and and height from address 24 and 26.

predefined type	byte: offset		field name
	20:	integer	style
gm_$grid_attributes_t	22:	integer	attributes
	24:		cross_size.x
gm_$point16_t	26:		cross_size.y

If the value of style is gm_$grid_box or gm_$grid_axis, insert or retrieve "prepeat" (line repetition factor) at address 24, "plength" (length of line pattern) at address 26, and "pattern" (line pattern) at byte addresses 28 - 35.

predefined type	byte: offset		field name
	20:	integer	style
gm_$grid_attributes_t	22:	integer	attributes
	24:	integer	prepeat
gm_$draw_pattern_t	26:	integer	plength
	28:		
	30:		pattern
	32:		
	34:		

GM_$HIGHLIGHT_T	A 2-byte integer. Specifies the type of highlighting. One of the following predefined values: GM_$OUTLINE Only value: Highlighting appears as an outline.
GM_$KEYSET_T	Specifies the set of characters that make up a keyset associated with the graphics input event types GM_$KEYSTROKE and GM_$BUTTONS. This is a 16-element array of 2-byte integers. For a FORTRAN subroutine to use in building a set of characters, see the routine GM_$INPUT_ENABLE in this volume.
GM_$LINE_STYLE_T	A 2-byte integer. Specifies the type of lines. One of the following predefined values: GM_$SAME_LINE_STYLE The line style does not change. GM_$SOLID The line style is solid. GM_$DOTTED The line style is dotted. GM_$PATTERNED The line style is patterned.
GM_$MODELCMD_MODE_T	A 2-byte integer. Specifies an editing mode for modeling commands. One of the following predefined values: GM_$MODELCMD_INSERT Modeling commands insert a command at the

current position in the currently open segment. This is equivalent to GM_$REPLACE_SET_FLAG = false.

GM_$MODELCMD_REPLACE
Modeling commands replace the command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_SET_FLAG = true.

GM_$MODELCMD_RUBBERBAND
Modeling commands XOR the previous modeling command on the screen, thus erasing it, then XOR the given modeling command onto the screen. Only bitplane 0 is used for rubberbanding. No changes are made to the metafile in this mode.

GM_$PLANE_LIST_T

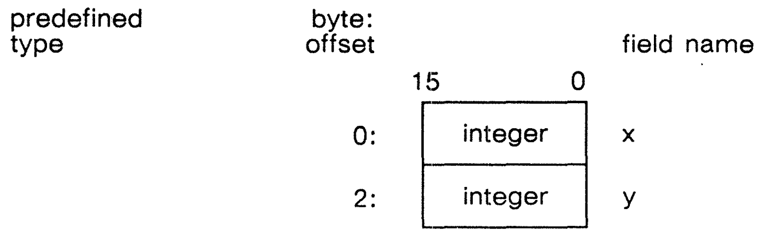
A 2-byte integer. Specifies a value between 0 and gm_$max_plane_id inclusive, depending on the type of node.

GM_$PLANE_MASK_T

A 2-byte integer. Specifies a set of planes from GM_$PLANE_LIST_T.

GM_$POINT16_T

Specifies the X- and Y- coordinates of a point. The diagram below illustrates the gm_$point16_t data type:



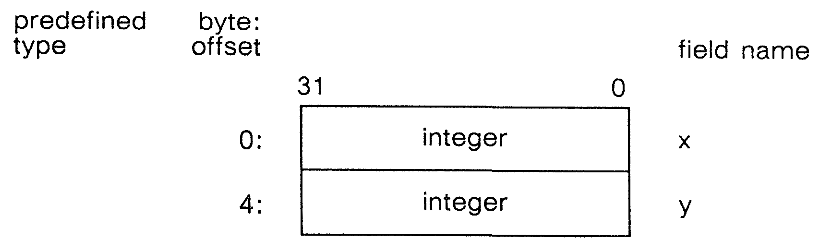
Field Description:

x
The x-coordinate of the point.

y
The y-coordinate of the point.

GM_$POINT32_T

Specifies the X- and Y-coordinates of a point. The diagram below illustrates the gm_$point32_t data type:



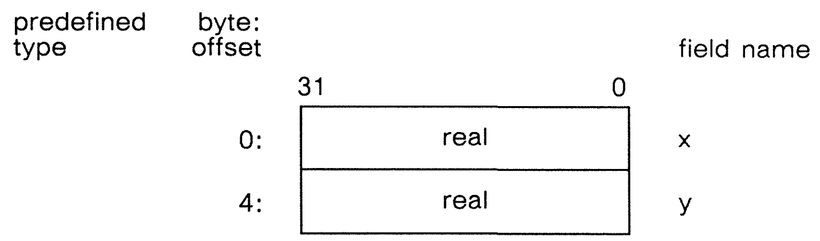
Field Description:

x
The x-coordinate of the point.

y
The y-coordinate of the point.

GM_$POINTREAL_T

Specifies the X- and Y-coordinates of a point. The diagram below illustrates the gm_$pointreal_t data type:



Field Description:

x
The x-coordinate of the point.

y
The y-coordinate of the point.

GM_$POINT_ARRAY16_T

An array of GM_$POINT16_T with MAX_ARRAY_LENGTH elements. The diagram for GM_$POINT_16_T illustrates a single element.

GM_$POINT_ARRAY32_T

An array of GM_$POINT32_T with MAX_ARRAY_LENGTH elements. The diagram for GM_$POINT32_T illustrates a single element.

GM_$POINT_ARRAYREAL_T

An array of GM_$POINTREAL_T with MAX_ARRAY_LENGTH elements. The diagram for GM_$POINTREAL_T illustrates a single element.

GM_$PRIMITIVE_PTR_T

Pointer to procedure for user-defined primitive, with the following argument protocol:

```
in N_POINTS          2-byte integer
in POINTS            array of
                     GM_$POINT16_T
in N_PARAMETERS      2-byte integer
in PARAMETERS        array of
                     GM_$POINTREAL_T
out STATUS           status_$T
```

GM_$PRINT_STYLE_T

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

GM_$GMF
Output is a graphics map file.

GM_$OUT1
Output is a vector command file.

GM_$POSTSCRIPT
Output file is a PostScript file.

GM_$REFRESH_PTR_T

Pointer to procedure for refreshing windows, with the following argument protocol:

```
in UNOBSCURED : boolean
in POS_CHANGE : boolean
```

GM_$ROTATE_REAL2X2_T

Specifies x- and y-coordinates for rotation. The diagram below illustrates the gm_$rotate_real2x2_t data type:

predefined type	byte: offset	31 0	field name
	0:	real	xx
	4:	real	xy
	8:	real	yx
	12:	real	yy

Field Description:

xx
The xx-coordinates for rotation.

xy
The xy-coordinates for rotation.

yx
The yx-coordinates for rotation.

yy
The yy-coordinates for rotation.

GM_$SEARCH_COMMAND_T

A 2-byte integer. Specifies the steps of a command search. One of the following predefined values:

GM_$CNEXT
Find the next command which falls within the pick aperture, moving forward in the segment.

GM_$STEP
Find the next command in the segment, independent of the pick aperture.

GM_$START
Move to the start of the segment, independent of the coordinates of the pick aperture.

GM_$END
Move to the end of the segment, independent of the coordinates of the pick aperture.

GM_$SEARCH_SEGMENT_T

A 2-byte integer. Specifies the steps of a segment search. One of the following predefined values:

GM_$SETUP
Make the top segment of the current viewport

the start of the list of picked segments. The rest of the list is emptied.

GM_$DOWN
Find the first segment instanced by the current segment, which when instanced falls within the pick aperture.

GM_$NEXT
Find the next segment within the segment one higher in the list of picked segments, which falls within the pick aperture.

GM_$UP
Move up one level in the list of picked segments.

GM_$TOP
Proceed to top segment in the list of picked segments, destroying the rest of the list of picked segments.

GM_$CLEAR
Clear the entire list of picked segments, allowing all segments to be edited or deleted.

GM_$BOTTOM
Perform GM_$DOWN repeatedly until a segment is reached for which GM_$DOWN finds nothing.

GM_$NEXTBOTTOM
Perform GM_$BOTTOM. If nothing is found, perform GM_$NEXT until a segment is found. An alternative to GM_$NEXT is one or more uses of GM_$UP followed by a GM_$NEXT. When a GM_$NEXT finds a segment, perform a GM_$BOTTOM from there.

GM_$SEGMENT_ID_T	A 4-byte integer. Specifies a value between 0 and gm_$max_segment_id inclusive.
GM_$SEGMENT_ID_LIST_T	Specifies an array of GM_$SEGMENT_ID_T with MAX_ARRAY_LENGTH elements.
GM_$STRING_T	An array of up to 256 characters. Specifies a string of characters.
GM_$VIEW_REFRESH_T	A 2-byte integer. Specifies the refresh state of the viewport. One of the following predefined values:

GM_$REFRESH_INHIBIT
When you change commands in the file, the viewport is rewritten when you call

GM_$VIEWPORT_REFRESH. GM_$DISPLAY_REFRESH does not affect a viewport in this refresh state. Other conditions are mode-dependent.

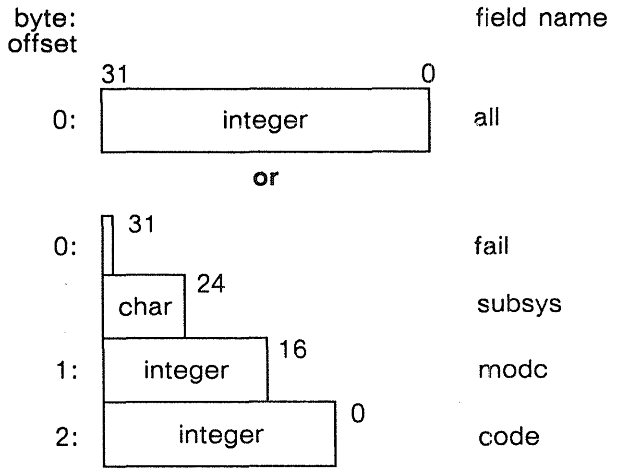
GM_$REFRESH_WAIT
When you change commands in the file, the viewport is rewritten when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH. Other conditions are mode-dependent.

GM_$REFRESH_UPDATE
Every time you change any command in the file, this viewport is completely corrected if it is the current viewport.

GM_$REFRESH_PARTIAL
Every time you change any command in the file, the following occurs if this viewport is the current viewport: Inserted primitive commands are added, and deleted primitive commands are erased, but underlying data is not rewritten.

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
2D GMR 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:

access The access mode, in GM_$ACC_CREATE_T format. This parameter is a 2-byte integer. Specify only one of the following predefined values:

GM_$WRITE	If the file already exists, an error code is returned in the status parameter.
GM_$OVERWRITE	If the file already exists, the previous version is deleted.
GM_$UPDATE	If the file already exists, the previous version is opened.

GM_$ABLOCK_ASSIGN_DISPLAY

Assigns an attribute block (by number) to an attribute class, for the entire display.

FORMAT

```
GM_$ABLOCK_ASSIGN_DISPLAY (aclass_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class to which the attribute block will be assigned. This is a 2-byte integer.

ablock_id
The identification number of the attribute block to be assigned to the attribute class. This is a 2-byte integer.

To assign the default attributes to an attribute class for the display, use ablock_id = 1.

To undo the asssignment of an attribute block to an attribute class for the display, use ablock_id = 0.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_ASSIGN_DISPLAY to assign an existing attribute block to an attribute class for all viewports in the display.

Use GM_$ABLOCK_INQ_ASSIGN_DISPLAY to inquire about the current attribute block number assigned to a particular class for the display.

Assignments of attribute blocks to attribute classes for individual viewports using GM_$ABLOCK_ASSIGN_VIEWPORT will override assignments made by GM_$ABLOCK_ASSIGN_DISPLAY.

GM_$ABLOCK_ASSIGN_VIEWPORT

Assigns an attribute block (by number) to an attribute class, for one viewport.

FORMAT

```
GM_$ABLOCK_ASSIGN_VIEWPORT (aclass_id, viewport_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class to which the attribute block will be assigned. This is a 2-byte integer.

To assign the default attributes to an attribute class for one viewport, use ablock_id = 1.

viewport_id
The identification number of the viewport in which to assign the attribute block to the attribute class. This is a 2-byte integer.

ablock_id
The identification number of the attribute block to be assigned to the attribute class. This is a 2-byte integer.

To undo the asssignment of an attribute block to an attribute class for one viewport, use ablock_id = 0.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_ASSIGN_VIEWPORT to assign an existing attribute block to an attribute class for one viewport in the display.

Use GM_$ABLOCK_INQ_ASSIGN_VIEWPORT to inquire about the current attribute block number assigned to a particular class for a particular viewport.

Assignments of attribute blocks to attribute classes for individual viewports using GM_$ABLOCK_ASSIGN_VIEWPORT will override assignments made by GM_$ABLOCK_ASSIGN_DISPLAY.

GM_$ABLOCK_COPY

Copies all attributes from one existing attribute block to another.

FORMAT

```
GM_$ABLOCK_COPY (source_ablock_id, destination_ablock_id, status)
```

INPUT PARAMETERS

source_ablock_id
The identification number of the existing attribute block from which attributes will be copied. This is a 2-byte integer.

destination_ablock_id
The identification number of the existing attribute block to which the attributes of the attribute block source_block_id will be copied. This is a 2-byte integer.

You may not copy attributes into attribute blocks 0 and 1 (default). Attribute block 0 is a list of no-change attribute values; attribute block 1 is a list of default attribute values.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_CREATE to establish a new attribute block identical to an existing one. Use GM_$ABLOCK_COPY to copy attributes from an existing attribute block to an existing one.

GM_$ABLOCK_CREATE

Creates an attribute block and initializes it equivalent to an existing block.

FORMAT

```
GM_$ABLOCK_CREATE (source_ablock_id, ablock_id, status)
```

INPUT PARAMETERS

source_ablock_id
The identification number of the existing attribute block used as the source for the block generated with GM_$ABLOCK_CREATE. This is a 2-byte integer.

OUTPUT PARAMETERS

ablock_id
The identification number assigned to the attribute block generated by GM_$ABLOCK_CREATE. This is a 2-byte integer.

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

USAGE

Use GM_$ABLOCK_CREATE to establish a new attribute block identical to an existing one. Use GM_$ABLOCK_COPY to copy attributes from an existing attribute block to an existing one.

Currently, you are limited to 10 attribute blocks, including the two preassigned ones.

GM_$ABLOCK_INQ_ASSIGN_DISPLAY

Returns the current attribute block number assigned to a particular attribute class for the display.

FORMAT

```
GM_$ABLOCK_INQ_ASSIGN_DISPLAY (aclass_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id

The identification number of the attribute class for which to return the current attribute block assignment. This is a 2-byte integer.

OUTPUT PARAMETERS

ablock_id

The identification number of the attribute block currently assigned to the specified attribute class for the display. This is a 2-byte integer.

If you have not assigned an attribute block to the specified attribute class for the display, the returned value is 0 (no assignment).

status

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

USAGE

Use GM_$ABLOCK_SET_ASSIGN_DISPLAY to assign an attribute block to a display.

GM_$ABLOCK_INQ_ASSIGN_VIEWPORT

Returns the current attribute block number assigned to a particular attribute class for one viewport.

FORMAT

```
GM_$ABLOCK_INQ_ASSIGN_VIEWPORT (aclass_id, viewport_id, ablock_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class for which to return the current attribute block assignment. This is a 2-byte integer.

viewport_id
The identification number of the viewport for which to return the current attribute block identification number. This is a 2-byte integer.

OUTPUT PARAMETERS

ablock_id
The identification number of the attribute block assigned to the attribute class for the display. This is a 2-byte integer.

If you have not assigned an attribute block to the special attribute class for the display, the returned value is 0 (no assignment).

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

USAGE

Use GM_$ABLOCK_SET_ASSIGN_VIEWPORT to assign an attribute block to a viewport.

GM_$ABLOCK_INQ_DRAW_RASTER_OP

Returns the raster operation code for drawing lines for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_DRAW_RASTER_OP (ablock_id, raster_op, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the raster operation codes. This is a 2-byte integer.

OUTPUT PARAMETERS

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15, or -1. The default value is 3. This sets all destination bit values to source bit values.

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

USAGE

Use GM_$ABLOCK_SET_DRAW_RASTER_OP to change the draw raster operation code in an attribute block.

GM_$ABLOCK_INQ_DRAW_STYLE

Returns the line style set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_DRAW_STYLE (ablock_id, style, repeat_factor, pattern,
                           pattern_length, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the drawing style. This is a 2-byte integer.

OUTPUT PARAMETERS

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

GM_$SOLID	Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default draw style is GM_$SOLID.
GM_$DOTTED	Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.
GM_$PATTERNED	Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.
GM_$SAME_DRAW_STYLE	Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor
The number of times each bit in this pattern is replicated before proceeding to the next bit in the pattern. This is a 2-byte integer. The replication factor changes the scaling applied to the pattern.

pattern
The bit pattern, in GM_$DRAW_PATTERN_T format. This is an array of 8 bytes constituting a 64-bit pattern. Only the bits specified in the pattern-length parameter are used.

pattern_length
The length of the bit pattern, in bits. This is a 2-byte integer. The returned values range from 1 to 64.

status

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

USAGE

Use GM_$ABLOCK_SET_DRAW_STYLE to change the line style in an attribute block.

GM_$ABLOCK_INQ_DRAW_VALUE

Returns the value for drawing lines set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_DRAW_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the drawing value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The line drawing value. This is a 4-byte integer. The default draw value is 1.

A value of -1 means that when this attribute block is selected, the draw value is not to be changed.

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

USAGE

Use GM_$ABLOCK_SET_DRAW_VALUE to change the line drawing value in an attribute block. The effect is influenced by the plane mask and the raster op.

GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE

Returns the background value for filling areas in the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the fill background value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The fill background value of the specified attribute block. This is a 4-byte integer. The default value is -2, the same as the viewport background.

The value -1 means that fill background pixels are to be left unchanged; that is, the fill background is "transparent."

The value -3 means that when this attribute block is selected, the fill background value is not to be changed.

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                   |
12345678901234567890123456789012   | 34567890
-----------------------------------|---------
GM_$ABLOCK_INQ_FILL_BACKGROUND_V   | ALUE
```

Use GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE to change the fill value in an attribute block.

GM_$ABLOCK_INQ_FILL_PATTERN

Returns the pattern set for filling areas for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FILL_PATTERN (ablock_id, scale, size, pattern, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the fill pattern. This is a 2-byte integer.

OUTPUT PARAMETERS

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

size
The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, these values must both be 32.

pattern
The 32x32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default pattern is all ones.

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

USAGE

Use GM_$ABLOCK_SET_FILL_PATTERN to change the fill pattern in an attribute block.

GM_$ABLOCK_INQ_FILL_VALUE

Returns the value set for filling areas for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FILL_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablockid
The identification number of the attribute block for which to return the fill value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The value for filling areas. This is a 4-byte integer. The default fill value is 1.

A value of -1 indicates that when this attribute block is selected, the fill value is not to be changed.

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

USAGE

Use GM_$ABLOCK_SET_FILL_VALUE to change the fill value in an attribute block.

GM_$ABLOCK_INQ_FONT_FAMILY

Returns the font family identification number set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_FONT_FAMILY (ablock_id, font_family_id, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text font family. This is a 2-byte integer.

OUTPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer. The default value is 1.

A value of -1 indicates that when this attribute block is selected, the font family is not to be changed.

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

USAGE

Use GM_$ABLOCK_SET_FONT_FAMILY to change the text font family identification in this attribute block.

GM_$ABLOCK_INQ_PLANE_MASK

Returns the value of the plane mask set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_PLANE_MASK (ablock_id, change, mask, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the plane mask values. This is a 2-byte integer.

OUTPUT PARAMETERS

change
A Boolean (logical) variable that indicates whether the plane mask is to be changed when the specified attribute block is selected. When true, the plane mask is to be changed to "mask." A value of change = false means that when this attribute block is selected, the plane mask is not to be changed. In this case, the value of mask is undefined.

mask
The plane mask, specifying which planes are currently in use, in GM_$PLANE_MASK_T format. This is a 2-byte integer. This value may be any combination of the set of integer values from 0 to 7. Each integer corresponds to a plane in use. For example, if 0 and 7 are set, planes 0 and 7 are in use. The default is that all planes are in use and can be modified.

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

USAGE

Operations can occur only on the planes specified in the mask. A program can use this routine, for example, to perform drawing operations only into certain planes in the bitmap.

Use GM_$ABLOCK_SET_PLANE_MASK to set the plane mask in an attribute block.

GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE

Returns the text background value set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id

The identification number of the attribute block for which to return the text background value. This is a 2-byte integer.

OUTPUT PARAMETERS

value

The value to use for the text background in this attribute block. This is a 4-byte integer.

The default text background value is -2. This specifies that the viewport background value is used as the text background. For borrowed displays and main memory bitmaps, this is always 0.

A value from 0 to 255 means to use that value.

-1 means that text background pixels are to be left unchanged; that is, the text background is "transparent."

-3 means that when this attribute block is selected, the text background value is not to be changed.

status

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$ABLOCK_INQ_TEXT_BACKGROUND_V | ALUE
```

Use GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE to set the text background value in an attribute block.

GM_$ABLOCK_INQ_TEXT_SIZE

Returns the size of text set for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_TEXT_SIZE (ablock_id, size, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text size. This is a 2-byte integer.

OUTPUT PARAMETERS

size
The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value. The default text size is 10.0.

A value of -1 indicates that when this attribute block is selected, the text size is not to be changed.

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

USAGE

The choice of a font from a font family is based on the specified text size. The largest font in the font family that does not exceed the text size is used. The size of a font is defined as the largest ascender height of any character in the font; the descender is ignored.

Use GM_$ABLOCK_SET_TEXT_SIZE to set the text size in an attribute block.

GM_$ABLOCK_INQ_TEXT_VALUE

Returns the value for writing text for the specified attribute block.

FORMAT

```
GM_$ABLOCK_INQ_TEXT_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block for which to return the text value. This is a 2-byte integer.

OUTPUT PARAMETERS

value
The value to use for writing text. This is a 4-byte integer. The default text value is 1.

A value of -1 indicates that when this attribute block is selected, the text value is not to be changed.

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

USAGE

Use GM_$ABLOCK_SET_TEXT_VALUE to set the text value in an attribute block.

GM_$ABLOCK_SET_DRAW_RASTER_OP

Changes the raster operation code for drawing lines for this attribute block.

FORMAT

```
GM_$ABLOCK_SET_DRAW_RASTER_OP (ablock_id, raster_op, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the drawing style. This is a 2-byte integer.

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15. The default value is 3. This sets all destination bit values to source bit values.

Assigning the value = -1 means that when this attribute block is selected, the draw raster op value is not to be changed.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_INQ_DRAW_RASTER_OP to retrieve the current raster operations in an attribute block.

GM_$ABLOCK_SET_DRAW_STYLE

Changes the value of the line style in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_DRAW_STYLE (ablock_id, style, repeat_factor, pattern,
                            pattern_length, status)
```

INPUT PARAMETERS

ablock_id

The identification number of the attribute block in which to change the drawing style. This is a 2-byte integer.

style

The style of line, in GM_$LINE_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$SOLID	Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default draw style is GM_$SOLID.
GM_$DOTTED	Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.
GM_$PATTERNED	Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.
GM_$SAME_DRAW_STYLE	Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor

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. Currently, repeat_factor is ignored and assumed to be 1.

pattern

The bit pattern, in GM_$DRAW_PATTERN_T format. This is an array of 8 bytes constituting a 64-bit pattern. Only the first pattern_length bits are used.

pattern_length

The length of the bit pattern, in bits. This is a 2-byte integer. Currently, pattern_length is ignored and assumed to be 64.

OUTPUT PARAMETERS

status

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

USAGE

The following defines a line pattern with dashes and spaces, twelve and four pixels long, respectively:

```
pattern          : STATIC gm_$draw_pattern_t :=
    [ CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    ];
```

Use GM_$ABLOCK_INQ_DRAW_STYLE to retrieve the current line style.

GM_$ABLOCK_SET_DRAW_VALUE

Changes the value for drawing lines in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_DRAW_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the drawing value. This is a 2-byte integer.

value
The value to use in drawing lines. This is a 4-byte integer. The default value is 1.

Assigning the value = -1 means that when this attribute block is selected, the draw value is not to be changed.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_INQ_DRAW_VALUE to retrieve the current draw value in an attribute block.

GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE

Changes the background value for filling areas in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the fill background value. This is a 2-byte integer.

value
The fill background value to use in the specified attribute block. This is a 4-byte integer. The default value is -2, the same as the viewport background.

Assigning a value from 0 to 255 means to use that value.

Assigning a value of -1 means that fill background pixels are to be left unchanged; that is, the fill background is "transparent."

Assigning the value = -3 means that when this attribute block is selected, the fill background value is not to be changed.

OUTPUT PARAMETERS

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$ABLOCK_SET_FILL_BACKGROUND_V | ALUE
```

Use GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE to retrieve the current fill background value in an attribute block.

GM_$ABLOCK_SET_FILL_PATTERN

Changes the fill pattern in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FILL_PATTERN (ablock_id, scale, size, pattern, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the fill pattern. This is a 2-byte integer.

scale
The number of times each bit in this pattern is to be replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, this value must be 1 (when defining a pattern), 0 (when clearing a pattern), or -1 (when specifying "no change").

A value scale = 0 indicates that filled areas are to be filled with a solid color and that the pattern is to be ignored. In this case, the fill value is assigned to every pixel in the interior of the specified area.

Assigning the value scale = -1 means that when this attribute block is selected, the fill pattern is not to be changed.

size
The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, these values must both be 32. See the GM_$ Data Types section for more information.

pattern
The 32 x 32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default pattern is all ones.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_INQ_FILL_PATTERN to retrieve the current fill pattern in an attribute block.

GM_$ABLOCK_SET_FILL_VALUE

Changes the value for filling areas in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FILL_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the fill value. This is a 2-byte integer.

value
The value for filling areas in the specified attribute block. This is a 4-byte integer. The default value is 1.

Assigning the value = -1 means that when this attribute block is selected, the fill value is not to be changed.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_INQ_FILL_VALUE to retrieve the current fill value in an attribute block.

GM_$ABLOCK_SET_FONT_FAMILY

Changes the font family in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_FONT_FAMILY (ablock_id, font_family_id, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text font family. This is a 2-byte integer.

font_family_id
The identification number assigned to the font family. This is a 2-byte integer. The default text font family identification number is 1.

Assigning value = -1 means that when this attribute block is selected, the font family is not to be changed.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_INQ_FONT_FAMILY to retrieve the current text font family identification in an attribute block.

Use GM_$FONT_FAMILY_INQ_ID to retrieve the identification number of a font family for which you know the name.

GM_$ABLOCK_SET_PLANE_MASK

Changes the value of the plane mask in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_PLANE_MASK (ablock_id, change, mask, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the plane mask. This is a 2-byte integer.

change
A Boolean (logical) variable that indicates whether the plane mask is to be changed when the specified attribute block is selected. When change is set to true, the plane mask is to be changed to "mask". Assigning change = false means that when this attribute block is selected, the plane mask is not to be changed.

mask
The plane mask, specifying which planes to use, in GM_$PLANE_MASK_T format. This is a 2-byte integer.

The default value is [0...7], in GM_$PLANE_MASK_T format, or 255 when expressed as a 2-byte integer. The default is that all planes are in use and can be modified.

FORTRAN programmers should encode the plane mask in a 2-byte integer in the range of 0-255 (1 means plane 0 is on, 2 means plane 1 is on, 3 means planes 0 and 1 are on, 255 means planes 0 through 7 are on).

OUTPUT PARAMETERS

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

USAGE

Operations can occur only on the planes specified in the mask. A program can use this routine, for example, to perform drawing operations only into certain planes in the bitmap.

Use GM_$ABLOCK_INQ_PLANE_MASK to retrieve the current plane mask.

FORTRAN programmers might want to include the parameter definitions given below:

```
 integer*2
+    bit0,
+    bit1,
+    bit2,
+    bit3,
+    bit4,
+    bit5,
+    bit6,
+    bit7
 parameter (
+    bit0  16#0001,
+    bit1  16#0002,
+    bit2  16#0004,
+    bit3  16#0008,
+    bit4  16#0010,
+    bit5  16#0020,
+    bit6  16#0040,
+    bit7  16#0080)
```

```
Example:

In FORTRAN, to enable planes 2 and 5, use the following:

    CALL GM_$PLANE_MASK( bit2 + bit5, status )

In Pascal, to enable planes 2 and 5, use the following:

    GM_$PLANE_MASK( [ 2, 5 ], status )
```

GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE

Changes the background value for text in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE (ablock_id, value,status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text background value. This is a 2-byte integer.

value
The value to use for the text background in this attribute block. This is a 4-byte integer.

The default text background value is -2. This specifies that the viewport background value is used as the text background. For borrowed displays and main memory bitmaps, this is always 0.

Assigning a value from 0 to 255 means to use that value.

Assigning a value of -1 means that text background pixels are to be left unchanged; that is, the text background is "transparent."

Assigning the value = -3 means that when this attribute block is selected, the text background value is not to be changed.

OUTPUT PARAMETERS

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$ABLOCK_SET_TEXT_BACKGROUND_V | ALUE
```

Use GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE to retrieve the current text background value in an attribute block.

GM_$ABLOCK_SET_TEXT_SIZE

Changes the size of text in this attribute block.

FORMAT

```
GM_$ABLOCK_SET_TEXT_SIZE (ablock_id, size, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text size. This is a 2-byte integer.

size
The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value. The default text size is 10.0.

The value of -1 indicates that when this attribute block is selected, the text size is not to be changed.

OUTPUT PARAMETERS

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

USAGE

The choice of a font from a family is based on the specified text size. The largest font in the family that does not exceed this height is used. The size of a font is defined as the largest ascender height on any character in the font; descender sizes are ignored.

Use GM_$ABLOCK_INQ_TEXT_SIZE to retrieve the current text size in an attribute block.

GM_$ABLOCK_SET_TEXT_VALUE

Changes the value for writing text set for this attribute block.

FORMAT

```
GM_$ABLOCK_SET_TEXT_VALUE (ablock_id, value, status)
```

INPUT PARAMETERS

ablock_id
The identification number of the attribute block in which to change the text value. This is a 2-byte integer.

value
The value to use for writing text. This is a 4-byte integer. The default text value is 1.

Assigning the value = -1 means that when this attribute block is selected, the text value is not to be changed.

OUTPUT PARAMETERS

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

USAGE

Use GM_$ABLOCK_INQ_TEXT_VALUE to retrieve the current text value in an attribute block.

GM_$ACLASS

Inserts a command into the current segment: change to a different attribute class.

FORMAT

```
GM_$ACLASS (aclass_id, status)
```

INPUT PARAMETERS

aclass_id
The identification number of the attribute class to use. This is 2-byte integer.

The maximum number of attribute classes is 16.

OUTPUT PARAMETERS

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

GM_$CIRCLE_[16,32,REAL]

Inserts a command into the current segment: draw a circle.

FORMAT

```
GM_$CIRCLE_16 (center, radius, fill, status)

GM_$CIRCLE_32 (center, radius, fill, status)

GM_$CIRCLE_REAL (center, radius, fill, status)
```

INPUT PARAMETERS

center

The point that is the center of the circle. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INQ_CIRCLE_16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INQ_CIRCLE_32
GM_$POINTREAL	A two-element array of real values for GM_$INQ_CIRCLE_REAL

See the GM_$ Data Types section for more information.

radius

The radius of the circle, in the appropriate format:

A 2-byte integer for GM_$CIRCLE_16

A 4-byte integer for GM_$CIRCLE_32

A real value for GM_$CIRCLE_REAL

fill

A Boolean (logical) value which specifies whether to fill the circle.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_CIRCLE_[16,32,REAL] to retrieve the parameters of a circle command inserted by GM_$CIRCLE_[16,32,REAL].

Circles may be scaled, rotated, and/or reflected. However, when you apply a transform in which one axis is stretched more than another, you get a circle of undefined size, not a distorted circle.

Before supplying coordinate data to GM_$CIRCLE_REAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$COMMAND_DELETE

Deletes the current command.

FORMAT

```
GM_$COMMAND_DELETE (status)
```

OUTPUT PARAMETERS

status

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

USAGE

After you delete the current command, the command before it in the current segment becomes the current command.

Use GM_$PICK_COMMAND to change the current command.

GM_$COMMAND_INQ_BOUNDS

Returns the bounds of the current command in the current segment.

FORMAT

```
GM_$COMMAND_INQ_BOUNDS (bounds,status)
```

OUTPUT PARAMETERS

bounds
Bounds of the command in GM_$BOUNDSREAL_T format. This is a four-element array of real numbers. See the GM_$ Data Types section for more information.

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

USAGE

Use this call to obtain the bottom left-hand and top right-hand coordinates of the current command in the current segment.

Use GM_$SEGMENT_INQ_BOUNDS to obtain the bounds of a segment.

Use GM_$FILE_INQ_BOUNDS to obtain the bounds of the primary segment in a file.

GM_$COORD_BITMAP_TO_PIXEL_2D

Converts fraction of GM bitmap coordinates to pixel coordinates.

FORMAT

```
GM_$COORD_BITMAP_TO_PIXEL_2D (bitmap_position, pixel_position, status)
```

INPUT PARAMETERS

bitmap_position
The bitmap coordinates to be converted to pixel coordinates, expressed as an (x,y) pair in terms of a fraction of the GM bitmap in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

pixel_position
The converted pixel coordinates in the current bitmap, in GM_$POINT16_T format. This is a two-element array of 2-byte integer values. See the GM_$ Data Types section for more information.

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

USAGE

Use this call only in GM_$CURRENT_BITMAP mode, when using 2D GMR with DOMAIN/Dialogue. For information on the DOMAIN/Dialogue user interface software refer to the *The DOMAIN/Dialogue User's Guide.*

GM_$COORD_BITMAP_TO_SEG_2D

Converts fractionn of GM bitmap coordinates to segment coordinates.

FORMAT

```
GM_$COORD_BITMAP_TO_SEG_2D (bitmap_position, segment_position, status)
```

INPUT PARAMETERS

bitmap_position
The bitmap coordinates to be converted to segment coordinates, expressed as an (x,y) pair in terms of a fraction of the GM bitmap in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

segment_position
The converted segment coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

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

USAGE

This routine converts the bitmap coordinates to segment coordinates of the primary segment of the current viewport.

In within-GPR mode, use GM_$COORD_PIXEL_TO_SEG_2D.

GM_$COORD_PIXEL_TO_BITMAP_2D

Converts pixel coordinates to fraction of GM bitmap coordinates.

FORMAT

```
GM_$COORD_PIXEL_TO_BITMAP_2D (pixel_position, bitmap_position, status)
```

INPUT PARAMETERS

pixel_position
The pixel coordinates in the current bitmap to be converted to GM bitmap coordinates, in GM_$POINT16_T format. This is a two-element array of 2-byte integer values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

bitmap_position
The bitmap coordinates expressed as an (x,y) pair in terms of a fraction of the GM bitmap in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

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

USAGE

Use this call only in GM_$CURRENT_BITMAP mode, when using 2D GMR with DOMAIN/Dialogue. For information on the DOMAIN/Dialogue user interface software refer to the *The DOMAIN/Dialogue User's Guide.*

GM_$COORD_PIXEL_TO_SEG_2D

Converts GPR bitmap coordinates used in within-GPR mode to segment coordinates, using a specified transformation.

FORMAT

```
GM_$COORD_PIXEL_TO_SEG_2D (rotate, translate, pixel_position,
                           segment_position, status)
```

INPUT PARAMETERS

rotate

The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

pixel_position

The pixel coordinates to be converted to segment coordinates, expressed as an (x,y) pair, in GM_$POINT16_T format. This is a 2-byte integer array of two elements. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

segment_position

The converted segment coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status

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

USAGE

In modes other than within-GPR mode, use GM_$COORD_BITMAP_TO_SEG_2D.

GM_$COORD_SEG_TO_BITMAP_2D

Converts segment coordinates to bitmap coordinates.

FORMAT

```
GM_$COORD_SEG_TO_BITMAP_2D (segment_position, bitmap_position, status)
```

INPUT PARAMETERS

segment_position
The segment coordinates to be converted to bitmap coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

bitmap_position
The converted bitmap coordinates, expressed as an (x,y) pair in terms of a fraction of the GM bitmap, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

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

USAGE

This routine converts the segment coordinates of the primary segment in the current viewport to bitmap coordinates.

In within-GPR mode, use GM_$COORD_SEG_TO_PIXEL_2D.

GM_$COORD_SEG_TO_PIXEL_2D

Converts within-GPR segment coordinates to GPR bitmap coordinates, using a specified transformation.

FORMAT

```
GM_$COORD_SEG_TO_PIXEL_2D (rotate, translate, segment_position,
                           pixel_position, status)
```

INPUT PARAMETERS

question

rotate

The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

segment_position

The segment coordinates to be converted to pixel coordinates, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

pixel_position

The converted pixel coordinates expressed as an (x,y) pair, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

status

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

USAGE

In modes other than within-GPR mode, use GM_$COORD_SEG_TO_BITMAP_2D.

GM_$CURSOR_INQ_ACTIVE

Returns the status of the cursor: displayed or not displayed.

FORMAT

```
GM_$CURSOR_INQ_ACTIVE (active, status)
```

OUTPUT PARAMETERS

active
A Boolean (logical) value that indicates whether or not 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.

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

USAGE

Use GM_$CURSOR_SET_ACTIVE to change the display status of the cursor.

Use GM_$CURSOR_SET_PATTERN to change the pattern of the cursor.

Use GM_$CURSOR_SET_POSITION to change the position of the cursor.

GM_$CURSOR_INQ_PATTERN

Returns the type, pattern, and origin of the cursor.

FORMAT

```
GM_$CURSOR_INQ_PATTERN (style, pattern_size, pattern, origin, status)
```

OUTPUT PARAMETERS

style
The cursor style, in GM_$CURSOR_STYLE_T format. This is a 2-byte integer. Currently, the only valid value is GM_$BITMAP.

pattern_size
The size of the cursor pattern, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, neither coordinate size may exceed 16. See the GM_$ Data Types section for more information.

pattern
The cursor pattern, in GM_$CURSOR_PATTERN_T format. This is an array of (pattern_size.y) 2-byte integers. The length of the array is determined by the y value of pattern_size.

origin
The offset from the pixel at the upper left of the cursor to the pixel at the origin of the cursor, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

When the cursor is moved using GM_$CURSOR_SET_POSITION, the pixel that is the cursor's origin is placed at the specified location.

The first element (x) indicates the number of cursor pixels that will be displayed to the left of the specified cursor location. The second element (y) indicates the number of cursor pixels that will be displayed above the specified cursor location. Both numbers must be between 0 and 15; only the first four bits are considered.

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

USAGE

Use GM_$CURSOR_SET_PATTERN to change the pattern of the cursor.

Use GM_$CURSOR_SET_ACTIVE to change the display status of the cursor.

Use GM_$CURSOR_SET_POSITION to change the position of the cursor.

GM_$CURSOR_INQ_POSITION

Returns the position of the cursor.

FORMAT

```
GM_$CURSOR_INQ_POSITION (bitmap_position, status)
```

OUTPUT PARAMETERS

bitmap_position
The converted bitmap coordinates, expressed as an (x,y) pair in terms of fractions of the GM bitmap, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

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

USAGE

Use GM_$CURSOR_SET_POSITION to change the position of the cursor.

Use GM_$CURSOR_SET_PATTERN to change the pattern of the cursor.

Use GM_$CURSOR_SET_ACTIVE to change the display status of the cursor.

GM_$CURSOR_SET_ACTIVE

Specifies whether or not the cursor is displayed.

FORMAT

```
GM_$CURSOR_SET_ACTIVE (active, status)
```

INPUT PARAMETERS

active

A Boolean (logical) value that indicates whether or not 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.

The default value for active is false.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$CURSOR_INQ_ACTIVE to retrieve the display status of the cursor.

GM_$CURSOR_SET_PATTERN

Specifies a cursor pattern, type, and origin.

FORMAT

```
GM_$CURSOR_SET_PATTERN (style, pattern_size, pattern, origin, status)
```

INPUT PARAMETERS

style
The cursor style, in GM_$CURSOR_STYLE_T format. Currently, the only valid value is GM_$BITMAP.

pattern_size
The size of the cursor pattern, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, neither coordinate size may exceed 16. See the GM_$ Data Types section for more information.

pattern
The cursor pattern, in GM_$CURSOR_PATTERN_T format. This is an array of (pattern_size.y) 2-byte integers. The length of the array is determined by the y value of pattern_size.

The default cursor uses the standard Display Manager pattern.

origin
The offset from the pixel at the upper left of the cursor to the pixel at the origin of the cursor, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

When the cursor is moved using GM_$CURSOR_SET_POSITION, the pixel that is the cursor's origin is placed at the specified location.

The first element (x) indicates the number of cursor pixels that will be displayed to the left of the specified cursor location. The second element (y) indicates the number of cursor pixels that will be displayed above the specified cursor location. Both numbers must be between 0 and 15; only the first four bits are considered.

OUTPUT PARAMETERS

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

USAGE

The default value is the standard Display Manager pattern.

Use GM_$CURSOR_INQ_PATTERN to retrieve the current pattern of the cursor.

You must place a cursor pattern smaller than 16x16 in the high-order bits of the first words of the pattern:

```
VAR
{ note that a cursor pattern smaller than 16x16
  starts in the high order bits, and starts
  in word 1 of the array }

cursor_pattern1 : gm_$cursor_pattern_t
                := [16#8080,16#4100,16#2200,16#1400,
                    16#800,16#1400,16#2200,16#4100,16#8080];
cursor_size : gm_$point16_t := [9,9];
cursor_origin : gm_$point16_t := [4,4];

.
.
.

gm_$cursor_set_pattern(gm_$bitmap,cursor_size,
                       cursor_pattern1,cursor_origin, status);
```

GM_$CURSOR_SET_POSITION

Moves the cursor on the screen.

FORMAT

```
GM_$CURSOR_SET_POSITION (bitmap_position, status)
```

INPUT PARAMETERS

bitmap_position
The converted bitmap coordinates, expressed as an (x,y) pair in terms of fractions of the GM bitmap, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

Use GM_$CURSOR_INQ_POSITION to retrieve the current position of the cursor.

GM_$CURVE_2D[16,32,REAL]

Inserts a command into the current segment: draw a curve.

FORMAT

```
GM_$CURVE_2D16 (curve_type, n_points, point_array, n_parameters,
                parameter_array, status)

GM_$CURVE_2D32 (curve_type, n_points, point_array, n_parameters,
                parameter_array, status)

GM_$CURVE_2DREAL (curve_type, n_points, point_array, n_parameters,
                  parameter_array, status)
```

INPUT PARAMETERS

curve_type

The type of curve to be drawn, in GM_$CURVE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$ARC_3P	Specifies an arc to be drawn through three points (n_points) in the point array (point_array). The value for n_points must equal 3.
GM_$SPLINE_CUBIC_P	Specifies a smooth curve (parametric cubic spline) to be drawn through the specified number of point (n_points) in the point array (point_array).

n_points

The number of points in the list of points. This is a 2-byte integer.

point_array

A list of coordinate points, each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$CURVE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$CURVE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$CURVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters

The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array

A list of parameters. This is an array of real values.

OUTPUT PARAMETERS

status

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

USAGE

Currently, n_parameters and parameter_array are not used.

Use GM_$INQ_CURVE_2D[16,32,REAL] to retrieve the parameters of a curve command inserted by GM_$CURVE_2D[16,32,REAL].

Curves are limited to 1000 (GM_$MAX_ARRAY_LENGTH) points.

Before supplying coordinate data to GM_$CURVE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$DATA_COERCE_INQ_REAL

Returns the data type to which real coordinates are converted.

FORMAT

```
GM_$DATA_COERCE_INQ_REAL (data_type, status)
```

OUTPUT PARAMETERS

data_type
The form in which to store data, in GM_$DATA_TYPE_T format. This is a 2-byte integer. Data sent to the package as real variables can be stored in another form. Currently, the only valid value is GM_$32.

You must set the data type to GM_$32 because real data must be coerced to 32-bit data before it can be stored.

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

USAGE

Use GM_$DATA_COERCE_SET_REAL to force real variables that you send to the package to be stored in another form.

GM_$DATA_COERCE_SET_REAL

Specifies the data type to which subsequent real coordinates are converted.

FORMAT

```
GM_$DATA_COERCE_SET_REAL (data_type, status)
```

INPUT PARAMETERS

data_type
The form in which to store data, in GM_$DATA_TYPE_T format. This is a 2-byte integer. Data sent to the package as real variables can be stored in another form. Currently, the only valid value is GM_$32.

You must set the data type to GM_$32 because real data must be coerced to 32-bit data before it can be stored.

OUTPUT PARAMETERS

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

USAGE

Use GM_$DATA_COERCE_INQ_REAL to retrieve the data type to which real coordinate data is to be coerced.

Currently, supplying real coordinate data before calling this routine is an error.

GM_$DISPLAY_FILE

Displays the entire current file in the current viewport.

FORMAT

```
GM_$DISPLAY_FILE (status)
```

OUTPUT PARAMETERS

status

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

USAGE

This command changes the view transformation to a value which will cause the entire metafile to be displayed as follows: one of the two dimensions fills 95 percent of the current viewport, and the other dimension fills less than or equal to 95 percent of the current viewport.

Note that the GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_FILE_PART

Displays part of the current file in the current viewport.

FORMAT

```
GM_$DISPLAY_FILE_PART (bounds, status)
```

INPUT PARAMETERS

bounds
The part of the primary segment of this file to be displayed, in terms of segment coordinates. This is a four-element array of real numbers (xmin, ymin, xmax, ymax), in GM_$BOUNDSREAL_T format. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

This command sets the view transformation to a value which causes the specified part of the file to be displayed as follows: one of the two dimensions fills the viewport, and the other dimension does not overflow the viewport.

The GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_INQ_COLOR_MAP

Returns the values in the display color map.

FORMAT

```
GM_$DISPLAY_INQ_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of first color value entry to be read. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer. Valid values are:

2	For monochromatic displays
1 - 16	For color displays in 4-plane configuration
1 - 256	For color displays in 8-plane configuration

OUTPUT PARAMETERS

values
Color value entries, in GM_$COLOR_VECTOR_T format. This is an array of real values. The array must be at least (3 * n_entries) in length.

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

USAGE

Use GM_$DISPLAY_SET_COLOR_MAP to change the value of the display color map.

GM_$DISPLAY_REFRESH

Redisplays all uninhibited viewports of the display.

FORMAT

```
GM_$DISPLAY_REFRESH (status)
```

OUTPUT PARAMETERS

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

USAGE

Viewports that are in the GM_$REFRESH_INHIBIT refresh state are not displayed.

GM_$DISPLAY_SEGMENT

Displays the specified segment (and all called segments) in the current viewport.

FORMAT

```
GM_$DISPLAY_SEGMENT (segment_id, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to display, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

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

USAGE

This command changes the view transformation to a value which will cause the entire segment to be displayed as follows: one of the two dimensions fills 95 percent of the current viewport, and the other dimension fills less than or equal to 95 percent of the current viewport.

Use GM_$DISPLAY_FILE to display the entire file.

Note that the GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_SEGMENT_GPR_2D

In within-GPR mode, allows you to display a segment within a GPR bitmap.

FORMAT

```
GM_$DISPLAY_SEGMENT_GPR_2D (segment_id, rotate, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to display, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

rotate
The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate
An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

You must specify the transform which relates segment coordinates of your selected segment to display coordinates (GPR coordinates). This is specified as a 2x2 rotation to be applied to coordinates in the segment, then a translation to be applied after this rotation.

If you have put data into segments with y pointing up, you will have to insert negative values into your transform.

The attributes you are currently using in your current GPR attribute block are used, until modified by attribute commands in the file.

In direct mode, a program must acquire the display before calling GM_$DISPLAY_SEGMENT_GPR_2D. The graphics metafile package will not acquire the display.

Rotation: Use the following to display segment little_seg at (400,300), at triple size and rotated 50 degrees:

```
rotate.xx := 3.0 * cos(50.0 * 3.14159/180.0);
rotate.xy := 3.0 * sin(50.0 * 3.14159/180.0);
rotate.yx := -rotate.xy;
rotate.yy := rotate.xx;
rpoint.x := 400.0;
rpoint.y := 300.0;
GM_$DISPLAY_SEGMENT_GPR_2D(little_seg, rotate,
                           rpoint, status);
```

Distortion: Use the following to display segment distort_seg at (12.5, 14.5), with a scale of 1 in the x direction and a scale of 3 in the y direction, unrotated:

```
rotate.xx := 1.0;
rotate.xy := 0.0;
rotate.yx := 0.0;
rotate.yy := 3.0;
rpoint.x  := 12.5;
rpoint.y  := 14.5;
GM_$DISPLAY_SEGMENT_GPR_2D(distort_seg, rotate,
                           rpoint, status);
```

GM_$DISPLAY_SEGMENT_PART

Displays part of the specified segment (and all called segments) in the current viewport.

FORMAT

```
GM_$DISPLAY_SEGMENT_PART (segment_id, bounds, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to display, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

bounds
The part of this segment to be displayed, in terms of segment coordinates. This is a four-element array of real values (xmin, ymin, xmax, ymax), in GM_$BOUNDSREAL_T format. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

This command sets the view transformation to a value which causes the specified part of the segment to be displayed as follows: one of the two dimensions fills the viewport, and the other dimension does not overflow the viewport.

It is necessary that ymax be greater than ymin and that xmax be greater than xmin.

Note that the GM package clears the viewport before displaying a file or segment in the viewport. To display more than one segment in a viewport, you must build a new segment which contains an instance of each segment you wish to display. You then display that composite segment.

GM_$DISPLAY_SET_COLOR_MAP

Changes values in the display color map.

FORMAT

```
GM_$DISPLAY_SET_COLOR_MAP (start_index, n_entries, values, status)
```

INPUT PARAMETERS

start_index
Index of first color value entry to be read. This is a 4-byte integer.

n_entries
Number of entries. This is a 2-byte integer. Valid values are:

2	For monochromatic displays
1 - 16	For color displays in 4-plane configuration
1 - 256	For color displays in 8-plane configuration

values
Color value entries, in GM_$COLOR_VECTOR_T format. This is an array of real values. The array must be at least (3 * n_entries) in length.

OUTPUT PARAMETERS

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

USAGE

The GM package initializes the color map to 0 = black, 1 = white.

Use GM_$DISPLAY_INQ_COLOR_MAP to retrieve the value of the display color map.

GM_$DRAW_RASTER_OP

Inserts a command into the current segment: change the logical raster operations to be performed when drawing.

FORMAT

```
GM_$DRAW_RASTER_OP (raster_op, status)
```

INPUT PARAMETERS

raster_op

Raster operation code. This is a 2-byte integer. Possible values are 0 through 15.

The default raster op value is 3. This sets all destination bit values to source bit values.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_DRAW_RASTER_OP to retrieve the parameters of a raster op command inserted by GM_$DRAW_RASTER_OP.

GM_$DRAW_STYLE

Inserts a command into the current segment: set the line style (solid, dotted).

FORMAT

```
GM_$DRAW_STYLE (style, repeat_factor, pattern, pattern_length, status)
```

INPUT PARAMETERS

style

The style of line, in GM_$LINE_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$SOLID	Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default drawing style is GM_$SOLID.
GM_$DOTTED	Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.
GM_$PATTERNED	Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.
GM_$SAME_DRAW_STYLE	Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor

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. Currently, repeat_factor is ignored and assumed to be 1.

pattern

The bit pattern, in GM_$DRAW_PATTERN_T format. This is an 8-byte array constituting of a 64-bit pattern. Only the first pattern_length bits are used.

pattern_length

The length of the bit pattern, in bits. This is a 2-byte integer. Allowed values are 1 through 64. Currently, pattern_length is ignored and assumed to be 64.

OUTPUT PARAMETERS

status

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

USAGE

The following defines a line pattern with dashes and spaces, twelve and four pixels long, respectively:

```
pattern          : STATIC gm_$draw_pattern_t :=
    [ CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    , CHAR( 2#11111111 ), CHAR( 2#11110000 )
    ];
```

When a styled line is drawn, pixels along the path which are not in the pattern are not affected. In other words, the implicit draw background value is transparent.

Use GM_$INQ_DRAW_STYLE to retrieve the current line style.

GM_$DRAW_VALUE

Inserts a command into the current segment: set the value used when drawing lines.

FORMAT

```
GM_$DRAW_VALUE (value, status)
```

INPUT PARAMETERS

value

The value used in drawing lines. This is a 4-byte integer.

The default draw value is 1.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_DRAW_VALUE to retrieve the current draw value.

GM_$FILE_CLOSE

Closes the current file, saving revisions or not.

FORMAT

```
GM_$FILE_CLOSE (save, status)
```

INPUT PARAMETERS

save

A Boolean (logical) value that indicates whether to save revisions. Set to true to save revisions to the currently open segment; set to false not to save revisions.

Currently, save is always assumed to be true.

If a segment is open in this file, the segment is closed and then the file is closed. If no segment was open, save is ignored.

OUTPUT PARAMETERS

status

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

GM_$FILE_COMPACT

Creates a new compacted GM file.

FORMAT

```
GM_$FILE_COMPACT(name, name_length, status)
```

INPUT PARAMETERS

name
The pathname of the file in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

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

USAGE

GM_$FILE_COMPACT changes the name of the the input file from * to *.bak (deleting any existing *.bak) and then creates a new compacted GM file named *.

With GM_$FILE_COMPACT, you can develop a file compacting utility like the following:

```
PROGRAM compact;

@%NOLIST;
@%INCLUDE '/sys/ins/base.ins.pas';
@%INCLUDE '/sys/ins/gmr.ins.pas';
@%INCLUDE '/sys/ins/pfm.ins.pas';
@%LIST;

VAR

    name        : name_$pname_t;
    length      : INTEGER;
    size        : gm_$point16_t := [ 0, 0 ];
    status      : status_$t;

BEGIN

    WRITE( 'File name:  ' );
    READLN( name );

    length := LASTOF( name );
    WHILE ( name[ length ] = ' ' ) AND ( length > 0 )
    DO length := length - 1;

    GM_$INIT
        ( gm_$no_bitmap
        , 0
        , size
        , 1
        , status
        );
    IF status.all <> status_$ok
    THEN pfm_$error_trap( status );

    GM_$FILE_COMPACT
        ( name
        , length
        , status
        );
    IF status.all <> status_$ok
    THEN pfm_$error_trap( status );

    GM_$TERMINATE
        ( status
        );
    IF status.all <> status_$ok
    THEN pfm_$error_trap( status );

    END.
```

GM_$FILE_CREATE

Creates a new graphics metafile and makes it the current file.

FORMAT

```
GM_$FILE_CREATE (name, name_length, access, concurrency, file_id, status)
```

INPUT PARAMETERS

name
The pathname of the file in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

access
The access mode, in GM_$ACC_CREATE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$WRITE	If the file already exists, an error message is returned.
GM_$OVERWRITE	If the file already exists, the previous version is deleted.
GM_$UPDATE	If the file already exists, the previous version is opened.

concurrency
The concurrency mode, defining the number of concurrent users the file may have, in GM_$CONC_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$1W	N readers or 1 writer is permitted.
GM_$COWRITERS	More than 1 writer is permitted, but all users must be on the same node. Only one segment in the file may be open at a time, and only one writer may be writing to a segment at a time.

OUTPUT PARAMETERS

file_id
The identification number assigned to the file. This is a 2-byte integer.

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

USAGE

The GM_$UPDATE access mode of GM_$FILE_CREATE and the GM_$CWR access mode of GM_$FILE OPEN produce identical results.

GM_$FILE_INQ_BOUNDS

Returns the bounds of the primary segment of a file.

FORMAT

```
GM_$FILE_INQ_BOUNDS (bounds,status)
```

OUTPUT PARAMETERS

bounds
Bounds of the primary segment of the file in GM_$BOUNDSREAL_T format. This is a four-element array of real numbers. See the GM_$ Data Types section for more information.

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

USAGE

Use this routine to obtain the bottom left-hand and top right-hand coordinates of the current file.

Use GM_$SEGMENT_INQ_BOUNDS to obtain the boundary of the current segment.

Use GM_$COMMAND_INQ_BOUNDS to obtain the boundary of the current command.

GM_$FILE_INQ_PRIMARY_SEGMENT

Returns the segment number assumed to be the start of the current file.

FORMAT

```
GM_$FILE_INQ_PRIMARY_SEGMENT (segment_id, status)
```

OUTPUT PARAMETERS

segment_id
The number of the primary segment of the current file, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

The primary segment is assumed to be the start of the picture.

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

GM_$FILE_OPEN

Reopens an existing file and makes it the current file.

FORMAT

```
GM_$FILE_OPEN (name, name_length, access, concurrency, file_id, status)
```

INPUT PARAMETERS

name
The pathname of the file in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

access
The read/write accessibility, GM_$ACC_OPEN_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$WR	Read or write. In this access mode, it is an error to attempt to open a nonexistent file.
GM_$R	Read only. In this access mode, it is an error to attempt to open a nonexistent file.
GM_$CWR	Read or write; if file does not exist, create it. The GM_$UPDATE access mode of GM_$FILE_CREATE and the GM_$CWR access mode of GM_$FILE OPEN produce identical results.

concurrency
The concurrency mode, defining the number of concurrent users the file may have, in GM_$CONC_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$1W	N readers or 1 writer is permitted.
GM_$COWRITERS	More than 1 writer is permitted, but all users must be on the same node. In GM_$COWRITERS concurrency mode, only one segment in the file may be open at a time, and only one writer may be writing to a segment at a time.

OUTPUT PARAMETERS

file_id
The identification number assigned to the file. This is a 2-byte integer.

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

USAGE

In modes other than GM_$CWR, it is an error to attempt to open a nonexistent file.

GM_$FILE_SELECT

Makes the specified file the current file.

FORMAT

```
GM_$FILE_SELECT (file_id, status)
```

INPUT PARAMETERS

file_id
The identification number of the file which is to become the current file. This is a 2-byte integer.

A file identification number is assigned by the GM package when GM_$FILE_CREATE or GM_$FILE_OPEN is called.

OUTPUT PARAMETERS

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

USAGE

When a file is created or opened, it becomes the current file. After closing the current file, you must select any other open file before you can use it.

GM_$FILE_SET_PRIMARY_SEGMENT

Changes the segment number assumed to be the start of the current file.

FORMAT

```
GM_$FILE_SET_PRIMARY_SEGMENT (segment_id, status)
```

INPUT PARAMETERS

segment_id
The number of the primary segment of the current file, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

The primary segment is assumed to be the start of the picture.

OUTPUT PARAMETERS

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

GM_$FILL_BACKGROUND_VALUE

Inserts a command into the current segment: set the value used for pixels not in the fill pattern when filling an area.

FORMAT

```
GM_$FILL_BACKGROUND_VALUE (value, status)
```

INPUT PARAMETERS

value

The value used in filling areas. This a 4-byte integer.

The default value is -2. This sets the fill background value equal to the viewport background value.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_FILL_BACKGROUND_VALUE to retrieve the parameters of a fill background value command inserted by GM_$FILL_BACKGROUND_VALUE.

GM_$FILL_PATTERN

Inserts a command into the current segment: set the pattern used for the interior of filled areas.

FORMAT

```
GM_$FILL_PATTERN (scale, size, pattern, status)
```

INPUT PARAMETERS

scale

The number of times each bit in this pattern is to be replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, this value must be 1 (when defining a pattern) or 0 (when clearing a pattern).

A value scale = 0 indicates that filled areas are to be filled with a solid color and that the pattern is to be ignored. In this case, the fill value is assigned to every pixel in the interior of the specified area.

The default value is scale = 0 (solid fill).

size

The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a two-element array of 2-byte integers. Currently, these values must both be 32. See the GM_$ Data Types section for more information.

pattern

The 32x32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default fill pattern is all ones.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_FILL_PATTERN to retrieve the parameters of a fill pattern command inserted by GM_$FILL_PATTERN.

GM_$FILL_VALUE

Inserts a command into the current segment: set the value used when filling an area.

FORMAT

```
GM_$FILL_VALUE (value, status)
```

INPUT PARAMETERS

value
The value used in filling areas. This is a 4-byte integer.

The default fill value is 1.

OUTPUT PARAMETERS

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

USAGE

Use GM_$INQ_FILL_VALUE to retrieve the parameters of a fill value command inserted by GM_$FILL_VALUE.

GM_$FONT_FAMILY

Inserts a command into the current segment: set the font family used when writing text.

FORMAT

```
GM_$FONT_FAMILY (font_family_id, status)
```

INPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

The default font family ID is 1.

OUTPUT PARAMETERS

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

USAGE

As is characteristic of other attribute commands, this command specifies the font family to be used for subsequent text commands of the current segment and all segments instanced from the current segment.

Use GM_$INQ_FONT_FAMILY to get the value stored for the current GM_$FONT_FAMILY command.

GM_$FONT_FAMILY_EXCLUDE

Undoes the inclusion of a font family.

FORMAT

```
GM_$FONT_FAMILY_EXCLUDE (font_family_id, status)
```

INPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

OUTPUT PARAMETERS

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

USAGE

Attempting to exclude a font family which is referenced by a font family command (as generated by GM_$FONT_FAMILY) is an error.

GM_$FONT_FAMILY_INCLUDE

Specifies a font family to use in this metafile.

FORMAT

```
GM_$FONT_FAMILY_INCLUDE (pathname, pathname_length, font_type,
                         font_family_id, status)
```

INPUT PARAMETERS

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the pathname. This is a 2-byte integer.

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$PIXEL	A font described pixel by pixel, including all DOMAIN standard fonts.
GM_$STROKE	A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

OUTPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

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

USAGE

Use GM_$FONT_FAMILY_INQ_ID to retrieve the font family identification of a previously included font family.

Currently, you must include at least one font family before text commands will be displayed.

GM_$FONT_FAMILY_INQ_ID

Returns the identification number of a previously included font family.

FORMAT

```
GM_$FONT_FAMILY_INQ_ID (pathname, pathname_length, font_type,
                        font_family_id, status)
```

INPUT PARAMETERS

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$PIXEL	A font described pixel by pixel, including all DOMAIN standard fonts.
GM_$STROKE	A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

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

USAGE

Use GM_$FONT_FAMILY_INCLUDE to change a font family to use in this metafile.

GM_$FONT_FAMILY_INQ_NAME

Returns the font family name for the specified identification number of a previously included font family.

FORMAT

```
GM_$FONT_FAMILY_INQ_NAME (font_family_id, font_type, pathname,
                           pathname_length, maximum_length, status)
```

INPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer.

maximum_length
The size of the array in the pathname parameter. This is a 2-byte integer.

OUTPUT PARAMETERS

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Contains only one of the following predefined values:

GM_$PIXEL	A font described pixel by pixel, including all DOMAIN standard fonts.
GM_$STROKE	A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the pathname. This is a 2-byte integer.

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

USAGE

Use GM_$FONT_FAMILY_INCLUDE to change a font family to use in this metafile.

GM_$FONT_FAMILY_RENAME

Changes the font family file corresponding to this font family identification.

FORMAT

```
GM_$FONT_FAMILY_RENAME (font_family_id, pathname, pathname_length,
                        font_type, status)
```

INPUT PARAMETERS

font_family_id
The identification number previously assigned to a font family. This is a 2-byte integer.

pathname
The pathname of the font family file in NAME_$PNAME_T format. This is an array of up to 256 characters.

pathname_length
The number of characters in the new pathname. This is a 2-byte integer.

font_type
The type of font, in GM_$FONT_TYPE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$PIXEL	A font described pixel by pixel, including all DOMAIN standard fonts.
GM_$STROKE	A font defined by stroke font metafiles. The characters in a stroke font are usually made up of vectors.

OUTPUT PARAMETERS

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

GM_$INIT

Initializes the graphics metafile package and opens the display.

FORMAT

```
GM_$INIT (display_mode, unit, size, n_planes, status)
```

INPUT PARAMETERS

display_mode

One of six modes of operation. Graphics metafile routines can operate by borrowing the entire display, by using a Display Manager window, by creating a main memory bitmap but no display bitmap, by using an already initialized GPR bitmap, and by building a file without a main memory or display memory bitmap. The value is in GM_$DISPLAY_MODE_T format. Specify only one of the following predefined values:

GM_$BORROW	Uses the entire screen.
GM_$DIRECT	Displays within a Display Manager window.
GM_$MAIN_BITMAP	Displays within a bitmap allocated in main memory.
GM_$NO_BITMAP	Allows editing of files without display.
GM_$WITHIN_GPR	Displays the output of the metafile within a bitmap that you initialize using routines of the DOMAIN graphics primitives.
GM_$CURRENT_BITMAP	Use the current DOMAIN/Dialogue GPR bitmap.

unit

This parameter has three possible meanings as follows:

The display unit, if the display mode is GM_$BORROW. This is a 2-byte integer. Currently, the only valid display unit number for borrow-display mode is 1.

The stream identifier for the pad, if the display mode is GM_$DIRECT. Use STREAM_$ID_T format. This is a 2-byte integer.

Any value, such as zero, in GM_$MAIN_BITMAP, GM_$NO_BITMAP, GM_$CURRENT_BITMAP, or GM_$WITHIN_GPR modes.

size

The size of the bitmap, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. The first element is the bitmap width in pixels; the second element is the bitmap height in pixels. Each value may be any number between 1 and 4096 (limits are reduced to the display or window size if necessary). See the GM_$ Data Types section for more information. This parameter is ignored in GM_$NO_BITMAP, GM_$WITHIN_GPR, and GM_$CURRENT_BITMAP modes.

n_planes

The number of bitmap planes. This is a 2-byte integer. The following are valid values.

```
For display memory bitmaps:

1      For monochromatic displays
1 - 4  For color displays in one- or two-board configuration
1 - 8  For color displays in three-board configuration


For main memory bitmaps: 1 - 8 for all displays
```

OUTPUT PARAMETERS

status

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

USAGE

You can use the "unit" parameter to display metafiles in a window other than the window from which you executed your GM program:

```
VAR
   wndw : pad_$window_desc_t;
   instid,stid : stream_$id_t;
   bitmap_size : gm_$point16_t := [1024,1024];

BEGIN {program}

wndw.top := 0;
wndw.left := 0;
wndw.width := 300;
wndw.height := 300;

pad_$create_window ('',0, pad_$transcript, 1,
                     wndw, stid, st);
pad_$create ('',0,pad_$input,stid, pad_$bottom,
             [pad_$init_raw],5, instid, st);

            { The "unit" parameter is the stream id of the pad
              in which you want to display metafiles. }

gm_$init (gm_$direct,stid,bitmap_size,8,st);
```

The graphics metafile package has its own "clean-up" handler that terminates GM whenever faults are encountered. It is not necessary for an application to install its own fault handler for this purpose. In fact, an application-installed fault handler will not work because GM will no longer be initialized by the time the fault handler is called.

To use the mode GM_$WITHIN_GPR, you must initialize GPR before calling GM_$INIT. In this mode, you have full control of the screen, but you must handle viewports and input yourself using GPR or other routines. In this mode, these parameters are ignored: unit, size, and n_planes.

GM_$WITHIN_GPR is useful when you already have a user interface and want to use it rather than GM for viewing. GM_$WITHIN_GPR allows you to build sequences of commands using the GM routines which change the contents of a metafile. You can then display the file using GM_$DISPLAY_SEGMENT_GPR_2D. This is the only GM display routine you may use in this mode.

Use GM_$CURRENT_BITMAP mode in the display mode parameter with applications that use 2D GMR within DOMAIN/Dialogue. Do this when your application has already established a GPR bitmap (in the DOMAIN/Dialogue graphics area) and you then wish to use 2D GMR. If you are using DOMAIN/Dialogue with 2D GMR, then you may need to use GM_$COORD_BITMAP_TO_PIXEL_2D and GM_$COORD_PIXEL_TO_BITMAP_2D routines. For more information on the DOMAIN/Dialogue user interface refer to the *DOMAIN/Dialogue User's Guide.*

GM_$INPUT_DISABLE

Disables an input event type.

FORMAT

```
GM_$INPUT_DISABLE (event_type, status)
```

INPUT PARAMETERS

event_type

The input event type to be disabled, in GM_$EVENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$KEYSTROKE
: Returned when you type a keyboard character.

GM_$BUTTONS
: Returned when you press a button on the mouse or bitpad puck.

GM_$LOCATOR
: Returned when you move the mouse or bitpad puck, or the touchpad.

GM_$ENTERED_WINDOW
: Returned when the cursor enters a window in which the GM bitmap resides. Direct mode is required.

GM_$LEFT_WINDOW
: Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode is required.

GM_$LOCATOR_STOP
: Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INPUT_ENABLE to enable an input event type.

GM_$INPUT_ENABLE

Enables an input event type.

FORMAT

```
GM_$INPUT_ENABLE (event_type, key_set, status)
```

INPUT PARAMETERS

event_type
The event type to be disabled, in GM_$EVENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$KEYSTROKE	Returned when you type a keyboard character.
GM_$BUTTONS	Returned when you press a button on the mouse or bitpad puck.
GM_$LOCATOR	Returned when you move the mouse or bitpad puck, or the touchpad.
GM_$ENTERED_WINDOW	Returned when the cursor enters a window in which the GM bitmap resides. Direct mode is required.
GM_$LEFT_WINDOW	Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode is required.
GM_$LOCATOR_STOP	Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

key_set
The set of specifically enabled characters when the event type is GM_$KEYSTROKE or GM_$BUTTONS, in GM_$KEYSET_T format. This is an array of up to 256 characters.

OUTPUT PARAMETERS

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

USAGE

Use GM_$INPUT_DISABLE to disable an input event type.

The routines GM_$INPUT_ENABLE and GM_$INPUT_EVENT_WAIT may acquire the display (in direct mode only). GM_$INPUT_ENABLE will acquire the display when locator events are enabled; GM_$INPUT_EVENT_WAIT acquires the display before waiting for an event (and releases it after waiting).

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

```
BUILD_SET -- Builds a Pascal set of characters for FORTRAN users.


INPUT ARGUMENTS

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

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

OUTPUT ARGUMENTS

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


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

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

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

```
       PROGRAM build_set

       subroutine build_set(list,no_of_entries,returned_set)

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


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

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

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

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

GM_$INPUT_EVENT_WAIT

Checks for or waits until an occurrence of an enabled input event.

FORMAT

```
GM_$INPUT_EVENT_WAIT (wait, event_type, event_data, bitmap_position,
                      viewport_id, segment_position, status)
```

INPUT PARAMETERS

wait

A Boolean (logical) value that specifies when control returns to the calling program. Set to true to wait for an enabled event to occur; set to false to return control to the calling program immediately, whether or not an event has occurred.

OUTPUT PARAMETERS

event_type

The event type which occurred, in GM_$EVENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$KEYSTROKE
Returned when you type a keyboard character.

GM_$BUTTONS
Returned when you press a button on the mouse or bitpad puck.

GM_$LOCATOR
Returned when you move the mouse or bitpad puck, or the touchpad.

GM_$ENTERED_WINDOW
Returned when the cursor enters a window in which the GM bitmap resides. Direct mode is required.

GM_$LEFT_WINDOW
Returned when the cursor leaves a window in which the GM bitmap resides. Direct mode is required.

GM_$LOCATOR_STOP
Returned when you stop moving the mouse or bitpad puck, or stop using the touchpad.

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 is a character. This parameter is not modified for other events.

bitmap_position

The position in the display bitmap at which graphics input occurred, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

viewport_id

The identification number of the viewport in which the location "bitmap_position" is found. This is a 2-byte integer.

segment_position

The position at which graphics input occurred, converted to segment coordinates of the viewport primary segment, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status

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

USAGE

If the location "bitmap_position" is not in any viewport, viewport_id is zero and segment_position is undefined.

If the location "bitmap_position" is in a viewport which is not displaying, segment_position is undefined.

The routines GM_$INPUT_ENABLE and GM_$INPUT_EVENT_WAIT may acquire the display (in direct mode only). GM_$INPUT_ENABLE will acquire the display when locator events are enabled; GM_$INPUT_EVENT_WAIT acquires the display before waiting for an event (and releases it after waiting).

GM_$INQ_ACLASS

Returns the value stored for the current (GM_$ACLASS) command.

FORMAT

```
GM_$INQ_ACLASS (aclass_id, status)
```

OUTPUT PARAMETERS

aclass_id
The identification number of the attribute class to use. This is a 2-byte integer.

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

USAGE

Inquiring about a command that is not an GM_$ACLASS command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_BITMAP_SIZE

Returns the size of the GM bitmap in pixels.

FORMAT

```
GM_$INQ_BITMAP_SIZE (size, planes, status)
```

OUTPUT PARAMETERS

size

The size of the GM bitmap created when the GM package was initialized, in GM_$POINT16_T format. This is a two-element array of 2-byte integers. See the GM_$ Data Types section for more information.

In direct mode, this routine returns the size of the part of the Display Manager window in which the GM package was initialized, excluding the edges of the window reserved by the Display Manager.

In borrow mode, this is the size of the part of the borrowed display in which the GM package was initialized.

In main-bitmap mode, this is the size of the main memory bitmap which was created when the GM package was initialized.

planes

The number of planes in the GM bitmap created when the GM package was initialized. This is a 2-byte integer.

status

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

GM_$INQ_CIRCLE_[16,32,REAL]

Returns the values stored for the current (GM_$CIRCLE) command.

FORMAT

```
GM_$INQ_CIRCLE_16 (center, radius, fill, status)

GM_$INQ_CIRCLE_32 (center, radius, fill, status)

GM_$INQ_CIRCLE_REAL (center, radius, fill, status)
```

OUTPUT PARAMETERS

center
The point that is the center of the circle. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$INQ_CIRCLE_16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$INQ_CIRCLE_32

GM_$POINTREAL
A two-element array of real values for GM_$INQ_CIRCLE_REAL

See the GM_$ Data Types section for more information.

radius
The radius of the circle, in the appropriate format:

A 2-byte integer for GM_$INQ_CIRCLE_16

A 4-byte integer for GM_$INQ_CIRCLE_32

A real value for GM_$INQ_CIRCLE_REAL

fill
A Boolean (logical) value which specifies whether the circle is filled.

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

USAGE

The parameters are returned as they were supplied to the GM_$CIRCLE command that inserted this command into the metafile.

Use $GM_$INQ_COMMAND_TYPE to get the command type and the data type of this command.

Use $GM_$COMMAND_DELETE and GM_$CIRCLE_[16,32,REAL] to change the parameters of this command.

Inquiring about a command that is not a GM_$CIRCLE command results in an error.

Currently, you must use GM_$INQ_CIRCLE_16 if the stored data type is GM_$16; you must use GM_$INQ_CIRCLE_32 or _REAL if the stored data type is GM_$32.

GM_$INQ_COMMAND_TYPE

Returns the command type and the data type of the current command in the current segment.

FORMAT

```
GM_$INQ_COMMAND_TYPE  (command_type, data_type, status)
```

OUTPUT PARAMETERS

command_type
The type of command, in GM_$COMMAND_TYPE_T format. This is a 2-byte integer. One of the following predefined values is returned:

```
GM_$TACLASS
GM_$TCIRCLE_2D
GM_$TCURVE_2D
GM_$TDRAW_RASTER_OP
GM_$TDRAWSTYLE
GM_$TDRAWVALUE
GM_$TFILLBVALUE
GM_$TFILLPATTERN
GM_$TFILLVALUE
GM_$TFONTFAMILY
GM_$TINSTANCE_SCALE_2D
GM_$TINSTANCE_TRANS_2D
GM_$TINSTANCE_TRANSFORM_2D
GM_$TPLANEMASK
GM_$TPOLYLINE_2D
GM_$TPRIMITIVE_2D
GM_$TRECTANGLE
GM_$TTAG
GM_$TTEXT_2D
GM_$TTEXTBVALUE
GM_$TTEXTSIZE
GM_$TTEXTVALUE
```

data_type
The data storage type, in GM_$DATA_TYPE_T format. The possible values for this parameter are the following:

GM_$16	Data is stored as GM_$POINT16_T
GM_$32	Data is stored as GM_$POINT32_T

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

USAGE

Use GM_$INQ_POLYLINE, GM_$INQ_TEXT, and other similar commands with data storage types to get the parameters of the command.

Use GM_$SEGMENT_CREATE, GM_$SEGMENT_OPEN and GM_$SEGMENT_CLOSE to change the current segment. Use GM_$PICK_COMMAND to change the current command.

If the current command is the blank space at the start of the segment, as it is after GM_$PICK_COMMAND(GM_$START,STATUS), this routine returns a GM_$NO_CURRRENT_COMMAND error.

GM_$INQ_CONFIG

Returns the current configuration of the display device.

FORMAT

```
GM_$INQ_CONFIG (configuration, status)
```

OUTPUT PARAMETERS

configuration

Current display configuration, in GM_$DISPLAY_CONFIG_T format. This is a 2-byte integer. One of the following predefined values is returned:

Returned Value	Display Type
GM_$BW_800x1024	monochromatic portrait
GM_$BW_1024x800	monochromatic landscape
GM_$COLOR_1024x1024x4	color 1024 x 1024 (DN6xx) 2-board config
GM_$COLOR_1024x1024x8	color 1024 x 1024 (DN6xx) 3-board config
GM_$COLOR_1024x800x4	color 1024 x 800 (DN5xx) 2-board config
GM_$COLOR_1024x1024x8	color 1024 x 800 (DN5xx) 3-board config
GM_$COLOR1_1024X800X8	color 1024 x 800 (DN570) 2-board config
GM_$COLOR_1280X1024X8	color 1280 x 1024 (DN580) 2-board config
GM_$COLOR2_1024X800X4	color 1024 x 800 (DN3000) 1-board config
GM_$BW_1280X1024	monochromatic (DN3000) 1-board config

status

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

USAGE

GM_$INQ_CONFIG is the only GM routine call that is usable when the graphics metafile package is not initialized.

GM_$INQ_CURVE_2D[16,32,REAL]

Returns the values stored for the current (GM_$CURVE) command.

FORMAT

```
GM_$INQ_CURVE_2D16 (curve_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$INQ_CURVE_2D32 (curve_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$INQ_CURVE_2DREAL (curve_type, n_points, point_array, n_parameters,
                      parameter_array, status)
```

OUTPUT PARAMETERS

curve_type
The type of curve, in GM_$CURVE_T format. This is a 2-byte integer. One of the following values is returned:

GM_$ARC_3P	Specifies an arc to be drawn through three points (n_points) in the point array (point_array). The value for n_points must equal 3.
GM_$SPLINE_CUBIC_P	Specifies a smooth curve (parametric cubic spline) to be drawn through the specified number of point (n_points) in the point array (point_array).

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinate points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INQ_CURVE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INQ_CURVE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INQ_CURVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters. This is an array of reals.

status

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

USAGE

Currently, n_parameters and parameter_array are not used.

Inquiring about a command that is not a GM_$CURVE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Use GM_$CURVE_2D[16,32,REAL] to change the parameters of this command.

Currently, you must use GM_$INQ_CURVE_16 if the stored data type is GM_$16; you must use GM_$INQ_CURVE_32 or _REAL if the stored data type is GM_$32.

GM_$INQ_DRAW_RASTER_OP

Returns the values stored for the current (GM_$DRAW_RASTER_OP) command.

FORMAT

```
GM_$INQ_DRAW_RASTER_OP (raster_op, status)
```

OUTPUT PARAMETERS

raster_op
Raster operation code. This is a 2-byte integer. Possible values are 0 through 15. The default value is 3.

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

USAGE

Use GM_$DRAW_RASTER_OP to change the raster operation codes.

Inquiring about a command that is not a GM_$DRAW_RASTER_OP command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_DRAW_STYLE

Returns the values stored for the current (GM_$DRAW_STYLE) command.

FORMAT

```
GM_$INQ_DRAW_STYLE (style, repeat_factor, pattern, pattern_length, status)
```

OUTPUT PARAMETERS

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

GM_$SOLID	Default. Specifies a solid line. If style = GM_$SOLID, then repeat_factor, pattern, and pattern_length are ignored. The default draw style is solid.
GM_$DOTTED	Specifies a line drawn in dashes. If style = GM_$DOTTED, then pattern and pattern_length are ignored. The result is equivalent to a patterned style, where the pattern is assumed to be one bit on and one bit off; the pattern_length is assumed to be 2. The replication factor is used to change the scaling applied to this pattern.
GM_$PATTERNED	Specifies a patterned line, determined by repeat_factor, pattern, and pattern_length.
GM_$SAME_DRAW_STYLE	Specifies that when this attribute block is selected, the draw style is not to be changed.

repeat_factor
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. Currently, repeat_factor is ignored and assumed to be 1.

pattern
The bit pattern, in GM_$DRAW_PATTERN_T format. This is an 8-byte array constituting a 64-bit pattern. Only the first pattern_length bits are used.

pattern_length
The length of the pattern. This is a 2-byte integer. Currently, pattern_length is ignored and assumed to be 64.

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

USAGE

Use GM_$SET_DRAW_STYLE to change the line style.

Inquiring about a command that is not a GM_$DRAW_STYLE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_DRAW_VALUE

Returns the value stored for the current (GM_$DRAW_VALUE) command.

FORMAT

```
GM_$INQ_DRAW_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The value used in drawing lines. This is a 4-byte integer.

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

USAGE

Use GM_$DRAW_VALUE to change the line drawing value.

Inquiring about a command that is not a GM_$DRAW_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_FILL_BACKGROUND_VALUE

Returns the value stored for the current (GM_$FILL_BACKGROUND_VALUE) command.

FORMAT

```
GM_$INQ_FILL_BACKGROUND_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The fill background value used in this command. This is a 4-byte integer. The default value is -2, the same as the viewport background.

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

USAGE

Inquiring about a command that is not a GM_$FILL_BACKGROUND_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Use GM_$FILL_BACKGROUND_VALUE to change the fill background value.

GM_$INQ_FILL_PATTERN

Returns the value stored for the current (GM_$FILL_PATTERN) command.

FORMAT

```
GM_$INQ_FILL_PATTERN (scale, size, pattern, status)
```

OUTPUT PARAMETERS

scale
The number of times each bit in this pattern is replicated (in both x and y directions) before proceeding to the next bit in the pattern. This is a 2-byte integer. Currently, this value must be 1 (when defining a pattern) or 0 (when clearing a pattern).

size
The size of the bit pattern, in bits, in the x and y directions; in GM_$POINT16_T format. This is a 2-byte integer array of two elements. Currently, these values must both be 32. See the GM_$ Data Types section for more information.

pattern
The 32x32 bit pattern to use in filling areas. This is a 32-element array of 4-byte integers. Each 4-byte integer represents one horizontal line of the pattern, starting at the top of the display. The default pattern is all ones.

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

USAGE

Use GM_$FILL_PATTERN to change the fill pattern.

Inquiring about a command that is not a GM_$FILL_PATTERN command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_FILL_VALUE

Returns the value stored for the current (GM_$FILL_VALUE) command.

FORMAT

```
GM_$INQ_FILL_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The value used in filling areas. This is a 4-byte integer. The default fill value is 1.

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

USAGE

Use GM_$FILL_VALUE to change the fill value.

Inquiring about a command that is not a GM_$FILL_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_FONT_FAMILY

Returns the value stored for the current (GM_$FONT_FAMILY) command.

FORMAT

```
GM_$INQ_FONT_FAMILY (font_family_id, status)
```

OUTPUT PARAMETERS

font_family_id
The identification number assigned to the font family. This is a 2-byte integer. The default value is 1.

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

USAGE

Use GM_$FONT_FAMILY to change the font family,

Inquiring about a command that is not a GM_$FONT_FAMILY command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_INSTANCE_SCALE_2D[16,32,REAL]

Returns the value stored for the current (GM_$INSTANCE_SCALE_2D) command.

FORMAT

```
GM_$INQ_INSTANCE_SCALE_2D16 (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_SCALE_2D32 (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_SCALE_2DREAL (segment_id, scale, translate, status)
```

OUTPUT PARAMETERS

segment_id
The identification number of the segment to be instanced, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

scale
A real number indicating the scaling factor.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INSTANCE_SCALE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INSTANCE_SCALE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INSTANCE_SCALE_2DREAL

See the GM_$ Data Types section for more information.

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

USAGE

Scaling is performed before translation.

Use GM_$INSTANCE_SCALE_2D[16,32,REAL] to change the segment instanced and its scale and translation parameters.

Inquiring about a command that is not a GM_$INSTANCE_SCALE_2D command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Currently, you must use GM_$INQ_INSTANCE_SCALE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_INSTANCE_SCALE_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL]

Returns the value stored for the current (GM_$INSTANCE_TRANSFORM) command.

FORMAT

```
GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL] (segment_id, rotate, translate,
                                            status)
```

OUTPUT PARAMETERS

segment_id
The identification number of the segment to transform, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

rotate
The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of 4 real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate
An (x,y) pair indicating the amount of translation, in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INSTANCE_TRANSFORM_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INSTANCE_TRANSFORM_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INSTANCE_TRANSFORM_2DREAL

See the GM_$ Data Types section for more information.

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                  |
12345678901234567890123456789012  | 34567890
----------------------------------|---------
GM_$INQ_INSTANCE_TRANSFORM_2DREA  | L
```

Use GM_$INSTANCE_TRANSFORM_2D[16,32,REAL] to change the segment instanced and its translation and rotation parameters.

Inquiring about a command that is not a GM_$INSTANCE_TRANSFORM_2D command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Currently, you must use GM_$INQ_INSTANCE_TRANSFORM_2D16 if the stored data type is GM_$16; you must use GM_$INQ_INSTANCE_TRANSFORM_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_INSTANCE_TRANSLATE_2D[16,32,REAL]

Returns the value stored for the current (GM_$INSTANCE_TRANSLATE_2D) command.

FORMAT

```
GM_$INQ_INSTANCE_TRANSLATE_2D16 (segment_id, translate, status)

GM_$INQ_INSTANCE_TRANSLATE_2D32 (segment_id,  translate, status)

GM_$INQ_INSTANCE_TRANSLATE_2DREAL (segment_id, translate, status)
```

OUTPUT PARAMETERS

segment_id
The identification number of the segment to be instanced, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INSTANCE_TRANSLATE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INSTANCE_TRANSLATE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INSTANCE_TRANSLATE_2DREAL

See the GM_$ Data Types section for more information.

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$INQ_INSTANCE_TRANSLATE_2DREA | L
```

Use GM_$INSTANCE_TRANSLATE_2D[16,32,REAL] to change the segment instanced and its translation parameters.

Inquiring about a command that is not a GM_$INSTANCE_TRANSLATE_2D command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Currently, you must use GM_$INQ_INSTANCE_TRANSLATE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_INSTANCE_TRANSLATE_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_PLANE_MASK

Returns the value stored for the current (GM_$PLANE_MASK) command.

FORMAT

```
GM_$INQ_PLANE_MASK (mask, status)
```

OUTPUT PARAMETERS

mask
The plane mask, specifying which planes to use, in GM_$PLANE_MASK_T format. This is a 2-byte integer. (See the description under GM_$PLANE_MASK.)

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

USAGE

Use GM_$PLANE_MASK to set the plane mask.

Inquiring about a command that is not a GM_$PLANE_MASK command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_POLYLINE_2D[16,32,REAL]

Returns the values stored for the current (GM_$POLYLINE_2D) command.

FORMAT

```
GM_$INQ_POLYLINE_2D16 (n_points, point_array, close, fill, status)

GM_$INQ_POLYLINE_2D32 (n_points, point_array, close, fill, status)

GM_$INQ_POLYLINE_2DREAL (n_points, point_array, close, fill, status)
```

OUTPUT PARAMETERS

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinates of points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INQ_POLYLINE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INQ_POLYLINE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INQ_POLYLINE_2DREAL

See the GM_$ Data Types section for more information.

close
A Boolean (logical) value which specifies whether the first and last points are connected. Set the parameter to true to close the polygon. You must use close when you want to fill a polygon.

fill
A Boolean (logical) value which specifies whether to fill the polygon or not. Filled polygons must be closed. Set the parameter to true to fill the polygon; set it to false for an unfilled polygon.

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

USAGE

The parameters are returned as they were supplied to the command GM_$POLYLINE_[16,32,REAL] which inserted this command into the metafile.

Currently, you must use GM_$INQ_POLYLINE_16 if the stored data type is gm_$16; you must use GM_$INQ_POLYLINE_32 or _REAL if the stored data type is gm_$32.

Inquiring about a command that is not a GM_$POLYLINE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_PRIMITIVE_2D[16,32,REAL]

Returns the values stored for the current (GM_$PRIMITIVE) command.

FORMAT

```
GM_$INQ_PRIMITIVE_2D16  (primitive_type, n_points, point_array, n_parameters,
                         parameter_array, status)

GM_$INQ_PRIMITIVE_2D32 (primitive_type, n_points, point_array, n_parameters,
                        parameter_array, status)

GM_$INQ_PRIMITIVE_2DREAL (primitive_type, n_points, point_array, n_parameters,
                          parameter_array, status)
```

OUTPUT PARAMETERS

primitive_type
The user-defined type of primitive command. This is a 2-byte integer.

n_points
The number of points in the list of points. This is a 2-byte integer.

point_array
A list of coordinates of points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$PRIMITIVE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$PRIMITIVE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$PRIMITIVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters, in GM_$ARRAYREAL_T format. This is an array of real values.

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

USAGE

Use GM_$PRIMITIVE_2D[16,32,REAL] to change the current value of the primitive command.

Currently, you must use GM_$INQ_PRIMITIVE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_PRIMITIVE_2D32 or _REAL if the stored data type is GM_$32.

GM_$INQ_RECTANGLE_[16,32,REAL]

Returns the values stored for the current (GM_$RECTANGLE) command.

FORMAT

```
GM_$INQ_RECTANGLE_16 (point1, point2, fill, status)

GM_$INQ_RECTANGLE_32 (point1, point2, fill, status)

GM_$INQ_RECTANGLE_REAL (point1, point2, fill, status)
```

OUTPUT PARAMETERS

point1, point2

The coordinates of two diagonally opposite corners, each a pair (x,y) of integers in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$RECTANGLE_16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$RECTANGLE_32
GM_$POINTREAL	A two-element array of real values for GM_$RECTANGLE_REAL

See the GM_$ Data Types section for more information.

The GM package sorts rectangle coordinates before storing them. The returned parameter point1 will contain the smaller x value and the smaller y value, regardless of the order in which you supplied the data.

fill

A Boolean (logical) value which specifies whether to fill the rectangle or not. Set the parameter to true to fill the rectangle; set it to false for an unfilled rectangle.

status

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

USAGE

Use GM_$RECTANGLE_[16,32,REAL] to change the values for this command.

Use GM_$INQ_RECTANGLE_[16,32,REAL] to retrieve the parameters of a rectangle command inserted by GM_$RECTANGLE_[16,32,REAL].

Currently, you must use GM_$INQ_RECTANGLE_16 if the stored data type is GM_$16; you must use GM_$INQ_RECTANGLE_32 or _REAL if the stored data type is GM_$32.

GM_$INQ_TAG

Returns the value stored for the current (GM_$TAG) command.

FORMAT

```
GM_$INQ_TAG (string, string_length, status)
```

OUTPUT PARAMETERS

string
The text string stored, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length
The length of the string. This is a 2-byte integer.

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

USAGE

Use GM_$TAG to change the text string stored in this command.

GM_$INQ_TEXT_2D[16,32,REAL]

Returns the value stored for the current (GM_$TEXT_2D[16,32,REAL]) command.

FORMAT

```
GM_$INQ_TEXT_2D[16,32,REAL] (point, rotate, string, string_length, status)
```

OUTPUT PARAMETERS

point

The coordinates of the point at which to locate text. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$TEXT_2D16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$TEXT_2D32

GM_$POINTREAL
: A two-element array of real values for GM_$TEXT_2DREAL

See the GM_$ Data Types section for more information.

rotate

The angle at which this text string is to be written, in degrees. This is a real value. A value of 0.0 degrees indicates left to right text. Other values indicate clockwise rotation. For example, -90.0 degrees specifies bottom to top.

string

The text string to write, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length

The length of the string. This is a 2-byte integer.

status

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

USAGE

Use GM_$TEXT_2D[16,32,REAL] to change the text string.

Inquiring about a command that is not a GM_$TEXT_2D[16,32,REAL] command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

Use GM_$INQ_TEXT_2D[16,32,REAL] to retrieve the parameters of a text command inserted by GM_$TEXT_2D[16,32,REAL].

Currently, you must use GM_$INQ_TEXT_2D16 if the stored data type is GM_$16; you must use GM_$INQ_TEXT_2D32 or _2DREAL if the stored data type is GM_$32.

GM_$INQ_TEXT_BACKGROUND_VALUE

Returns the value stored for the current (GM_$TEXT_BACKGROUND_VALUE) command.

FORMAT

```
GM_$INQ_TEXT_BACKGROUND_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The background value to use when writing text. This is a 4-byte integer.

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

USAGE

Use GM_$TEXT_BACKGROUND_VALUE to change the text background value.

Inquiring about a command that is not a GM_$TEXT_BACKGROUND_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_TEXT_SIZE

Returns the value stored for the current (GM_$TEXT_SIZE) command.

FORMAT

```
GM_$INQ_TEXT_SIZE (size, status)
```

OUTPUT PARAMETERS

size
The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value.

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

USAGE

Use GM_$TEXT_SIZE to change the text size.

Inquiring about a command that is not a GM_$TEXT_SIZE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INQ_TEXT_VALUE

Returns the value stored for the current (GM_$TEXT_VALUE) command.

FORMAT

```
GM_$INQ_TEXT_VALUE (value, status)
```

OUTPUT PARAMETERS

value
The value to use when writing text. This is a 4-byte integer.

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

USAGE

Use GM_$TEXT_VALUE to change the text value.

Inquiring about a command that is not a GM_$TEXT_VALUE command results in an error.

You can use GM_$INQ_COMMAND_TYPE to determine the type of the current command.

GM_$INSTANCE_SCALE_2D[16,32,REAL]

Inserts a command into the current segment: instance the specified segment with the specified scale and translation parameters.

FORMAT

```
GM_$INSTANCE_SCALE_2D16 (segment_id, scale, translate, status)

GM_$INSTANCE_SCALE_2D32 (segment_id, scale, translate, status)

GM_$INSTANCE_SCALE_2DREAL (segment_id, scale, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to instance, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

scale
A real number indicating the scaling factor.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INSTANCE_SCALE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INSTANCE_SCALE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INSTANCE_SCALE_2DREAL

See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

Scaling is performed before translation.

Use GM_$SEGMENT_GET_ID to find the number of a previously defined segment for which you know only the name.

Use GM_$INQ_INSTANCE_SCALE_2D[16,32,REAL] to retrieve the parameters of an instance scale command inserted by GM_$INSTANCE_SCALE_[16,32,REAL].

Before supplying coordinate data to GM_$INSTANCE_SCALE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$INSTANCE_TRANSFORM_2D[16,32,REAL]

Inserts a command to instance the specified segment with the specified rotation and translation applied.

FORMAT

```
GM_$INSTANCE_TRANSFORM_2D16 (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSFORM_2D32 (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSFORM_2DREAL (segment_id, rotate, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to transform, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

rotate
The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INSTANCE_TRANSFORM_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INSTANCE_TRANSFORM_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INSTANCE_TRANSFORM_2DREAL

See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                    |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$INQ_INSTANCE_TRANSFORM_2DREA | L
```

Rotation is performed before translation.

Use GM_$SEGMENT_GET_ID to find the number of a previously defined segment for which you know only the name.

Use GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL] to retrieve the parameters of an instance transform command inserted by GM_$INSTANCE_TRANSFORM_2D[16,32,REAL].

You must call GM_$DATA_COERCE_SET_REAL before supplying coordinate data to GM_$INSTANCE_TRANSFORM_2DREAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

Rotation: Use the following to include an instance of segment little_seg at (400,300), at triple size and rotated 50 degrees:

```
rotate.xx := 3.0 * cos(50.0 * 3.14159/180.0);
rotate.xy := 3.0 * sin(50.0 * 3.14159/180.0);
rotate.yx := -rotate.xy;
rotate.yy := rotate.xx;
point.x  := 400;
point.y  := 300;
GM_$INSTANCE_TRANSFORM_2D16(little_seg, rotate, point, status);
```

Distortion: Use the following to include an instance of segment distort_seg at (12.5, 14.5), with a scale of 1 in the x direction and a scale of 3 in the y direction, unrotated:

```
rotate.xx := 1.0;
rotate.xy := 0.0;
rotate.yx := 0.0;
rotate.yy := 3.0;
rpoint.x  := 12.5;
rpoint.y  := 14.5;
GM_$INSTANCE_TRANSFORM_2DREAL(distort_seg, rotate, rpoint, status);
```

GM_$INSTANCE_TRANSLATE_2D[16,32,REAL]

Inserts a command into the current segment: instance the identified segment with the specified translation.

FORMAT

```
GM_$INSTANCE_TRANSLATE_2D16 (segment_id, translate, status)

GM_$INSTANCE_TRANSLATE_2D32 (segment_id, translate, status)

GM_$INSTANCE_TRANSLATE_2DREAL (segment_id, translate, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to instance, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

translate
An (x,y) pair indicating the amount of translation in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$INSTANCE_TRANSLATE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$INSTANCE_TRANSLATE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$INSTANCE_TRANSLATE_2DREAL

See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$INQ_INSTANCE_TRANSLATE_2DREA | L
```

Use GM_$INQ_INSTANCE_TRANSLATE_2D[16,32,REAL] to retrieve the parameters of an instance translate command inserted by GM_$INSTANCE_TRANSLATE_2D[16,32,REAL].

You must call GM_$DATA_COERCE_SET_REAL before supplying coordinate data to GM_$INSTANCE_TRANSLATE_2DREAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$MODELCMD_INQ_MODE

Returns the values stored for the current (GM_$MODELCMD_SET_MODE) command.

FORMAT

```
GM_$MODELCMD_INQ_MODE (gm_modelcmd_mode, status)
```

OUTPUT PARAMETERS

gm_modelcmd_mode
The editing mode, in GM_$MODELCMD_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$MODELCMD_INSERT
Modeling commands insert a command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = false.

GM_$MODELCMD_REPLACE
Modeling command replaces the command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = true.

GM_$MODELCMD_RUBBERBAND
Modeling commands XOR the previous command on the screen, thus erasing it, then XOR the given command onto the screen. Only bit plane 0 is used for rubberbanding. No changes are made to the metafile in this mode.

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

USAGE

Use GM_$MODELCMD_SET_MODE to set the model command mode.

GM_$MODELCMD_SET_MODE

Sets the modeling command mode.

FORMAT

```
GM_$MODELCMD_SET_MODE (gm_modelcmd_mode, status)
```

INPUT PARAMETERS

gm_modelcmd_mode

The editing mode to use, in GM_$MODELCMD_MODE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$MODELCMD_INSERT	Modeling commands insert a command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = false.
GM_$MODELCMD_REPLACE	Modeling commands replace the command at the current position in the currently open segment. This is equivalent to GM_$REPLACE_$SET_FLAG = true.
GM_$MODELCMD_RUBBERBAND	Modeling commands XOR the previous modeling command on the screen, thus erasing it, then XOR the given modeling command onto the screen. Only bitplane 0 is used for rubberbanding. No changes are made to the metafile in this mode.

OUTPUT PARAMETERS

status

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

USAGE

Rubberband mode provides an interactive capability. The mode uses a pointing device and allows an application program to get information from a user. The application program then uses the information to insert or replace the command with the changes the user has specified.

You may still set or inquire the replace flag by calling GM_$REPLACE_$SET_FLAG GM_$REPLACE_$INQ_FLAG, respectively. In new programs, use the GM_$MODELCMD... routines.

Use GM_$MODELCMD_INQ_MODE to get the values stored in this command.

Only primitive and instance command types may be replaced. The GM_$MODELCMD_REPLACE mode may only be used if the current command is a primitive or instance command.

GM_$PICK_COMMAND

Within the current segment, selects a command which contains a selected point on the display.

FORMAT

```
GM_$PICK_COMMAND (search_rule, status)
```

INPUT PARAMETERS

search_rule

The search rule to apply in selecting the command, in GM_$SEARCH_COMMAND_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$CNEXT	Find the next command which falls within the pick aperture, moving forward in the segment.
GM_$STEP	Find the next command in the segment, independent of the coordinates of the pick aperture.
GM_$START	Move to the start of the segment, independent of the coordinates of the pick aperture. If search_rule = GM_$START, the current command is changed to equal beginning-of-segment (no current command), allowing commands to be added at the beginning of the segment.
GM_$END	Move to the end of the segment, independent of the coordinates of the pick aperture.

OUTPUT PARAMETERS

status

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

USAGE

The current command is changed to equal the picked command.

Instance commands are treated like any other command in their context. To pick "into" an instanced segment, use GM_$PICK_SEGMENT.

Commands are picked only if the pick aperture intersects the drawn command. An exception is the GM_$CURVE_2D... commands. These commands are picked if the bounding box of the command intersects the pick aperture.

GM_$PICK_HIGHLIGHT_COMMAND

Highlights the current command on the display.

FORMAT

```
GM_$PICK_HIGHLIGHT_COMMAND (highlight, time, status)
```

INPUT PARAMETERS

highlight

The method to be used for highlighting the command, in GM_$HIGHLIGHT_T format. This is a 2-byte integer. Currently, the only possible value is GM_$OUTLINE. This value draws a rectangular outline around the command, leaves it displayed for the specified amount of time, and then erases it.

time

The number of seconds for which the command is to be highlighted. This is a real value.

OUTPUT PARAMETERS

status

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

USAGE

This operation is performed only if the viewport primary segment of the current viewport is the current segment.

The outline drawn around picked commands and picked segments is temporary. The outline remains on the screen for the requested number of seconds and is then erased.

GM_$PICK_HIGHLIGHT_SEGMENT

Within the current file, highlights the specified segment.

FORMAT

```
GM_$PICK_HIGHLIGHT_SEGMENT (highlight, time, status)
```

INPUT PARAMETERS

highlight
The method to be used for highlighting the segment, in GM_$HIGHLIGHT_T format. This is a 2-byte integer. Currently, the only possible value is GM_$OUTLINE. This value draws a rectangular outline around the segment, leaves it displayed for the specified amount of time, and then erases it.

time
The number of seconds for which the segment is to be highlighted. This is a real value.

OUTPUT PARAMETERS

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

USAGE

This operation is performed only if the viewport primary segment of the current viewport is the first segment in the pick list.

The outline drawn around picked segments and picked commands is temporary. The outline remains on the screen for the requested number of seconds and is then erased.

GM_$PICK_INQ_CENTER

Returns the center of the pick aperture.

FORMAT

```
GM_$PICK_INQ_CENTER (center, status)
```

OUTPUT PARAMETERS

center
The (x,y) coordinates of the center of the pick aperture, in GM_$POINTREAL_T format. This is a two-element array of real values.

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

USAGE

A program must call GM_$PICK_SET_CENTER and GM_$PICK_SET_SIZE after the use of the commands GM_$FILE_DISPLAY or GM_$SEGMENT_DISPLAY.

GM_$PICK_INQ_LIST

Returns the current list of picked segments.

FORMAT

```
GM_$PICK_INQ_LIST (max_length, length, list, status);
```

INPUT PARAMETERS

max_length
The maximum length of list you are prepared to receive. This is a 2-byte integer.

OUTPUT PARAMETERS

length
The number of segment ID's returned. This is a 2-byte integer.

list
An array of segment ID's, each in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

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

GM_$PICK_INQ_MASK

Returns the value of the mask used for segment pickable values during pick segment operations.

FORMAT

```
GM_$PICK_INQ_MASK (mask, status)
```

OUTPUT PARAMETERS

mask
The pick mask value. This is a 4-byte integer.

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

USAGE

Use GM_$PICK_SET_MASK to change the current value of the pick mask.

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$PICK_INQ_SIZE

Returns the size of the pick aperture.

FORMAT

```
GM_$PICK_SET_SIZE (size, status)
```

OUTPUT PARAMETERS

size

The x and y tolerances for the pick aperture, in segment coordinates of the current segment, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

status

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

USAGE

Use GM_$PICK_SET_SIZE to change the size of the pick aperture.

GM_$PICK_INQ_THRESHOLD

Returns the value of the threshold used for segment pickable values during pick segment operations.

FORMAT

```
GM_$PICK_INQ_THRESHOLD (threshold, status)
```

OUTPUT PARAMETERS

threshold
The pick threshold value. This is a 4-byte integer.

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

USAGE

Use GM_$PICK_SET_THRESHOLD to change the current value of the pick threshold.

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$PICK_SEGMENT

Selects a segment which contains a specified point on the display.

FORMAT

```
GM_$PICK_SEGMENT (search_rule, segment_id, n_instances, bounds, status)
```

INPUT PARAMETERS

search_rule
The search rule to apply in selecting the command, in GM_$SEARCH_SEGMENT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$SETUP	Make the top segment of the current viewport the start of the list of picked segments. The rest of the list is emptied.
GM_$DOWN	Find the first segment instanced by the current segment which, when instanced, falls within the pick aperture.
GM_$NEXT	Find the next segment within the segment one higher in the list of picked segments, which falls within the pick aperture.
GM_$UP	Move up one level in the list of picked segments.
GM_$TOP	Proceed to top segment in the list of picked segments, destroying the rest of the list of picked segments.
GM_$CLEAR	Clear the entire list of picked segments, allowing all segments to be edited or deleted.
GM_$BOTTOM	Perform GM_$DOWN repeatedly until a segment is reached for which GM_$DOWN finds nothing.
GM_$NEXTBOTTOM	Perform GM_$BOTTOM. If nothing is found, perform GM_$NEXT, or one or more GM_$UPs followed by a GM_$NEXT, until a GM_$NEXT finds a segment. When a GM_$NEXT finds a segment, perform a GM_$BOTTOM from there.

OUTPUT PARAMETERS

segment_id
The identification number of the picked segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

n_instances
The number of instances of this segment in the file. This is a 4-byte integer.

bounds
(Reserved for future extension.) A GM_$BOUNDSREAL_T variable. This is an array of 4 real values.

status

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

USAGE

Use GM_$PICK_SEGMENT to set the current segment.

If a segment is picked, the picked segment becomes the last segment on the list of picked segments. If no segment is picked, the list of picked segments is unchanged. While a segment is in the picked list, it may not be deleted or edited.

Use GM_$INQ_PICK_LIST to examine the current list of picked segments.

Each segment listed in the list of picked segments is instanced in the preceding segment.

GM_$PICK_SET_CENTER

Changes the center of the pick aperture.

FORMAT

```
GM_$PICK_SET_CENTER (center, status)
```

INPUT PARAMETERS

center

The (x,y) coordinates of the center of the pick aperture, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status

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

USAGE

When picking commands, GM_$PICK_SET_CENTER uses the segment coordinates of the current segment.

When picking segments, in other than no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the viewport primary segment of the segment in which pick segment operations were initialized.

When picking segments in no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the primary segment of the file.

The PICK routines search for any segments/commands which fall into the following region:

```
(center.x - size.x  to  center.x + size.x,
 center.y - size.y  to  center.y + size.y)
```

GM_$PICK_SET_MASK

Changes the value of the mask used for segment pickable values during pick segment operations.

FORMAT

```
GM_$PICK_SET_MASK (mask, status)
```

INPUT PARAMETERS

mask
The pick mask value. This is a 4-byte integer.

The pick mask is initialized to 16#7FFF (all segments with nonzero pickable values are pickable).

OUTPUT PARAMETERS

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

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

Use GM_$PICK_INQ_MASK to retrieve the current value of the pick mask.

GM_$PICK_SET_SIZE

Specifies the size of the pick aperture.

FORMAT

```
GM_$PICK_SET_SIZE (size, status)
```

INPUT PARAMETERS

size

The x and y tolerances for the pick aperture, in segment coordinates of the current segment, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$PICK_INQ_SIZE to retrieve the size of the pick aperture.

When picking commands, GM_$PICK_SET_CENTER uses the segment coordinates of the current segment.

When picking segments, in other than no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the viewport primary segment of the segment in which pick segment operations were initialized.

When picking segments, in no-bitmap mode, GM_$PICK_SET_CENTER uses the segment coordinates of the primary segment of the file.

The dimensions for the pick aperture are the following: (2 * size.x, 2 * size.y).

The PICK routines search for any segments/commands which fall into the following region:

```
(center.x - size.x  to  center.x + size.x,
 center.y - size.y  to  center.y + size.y).
```

GM_$PICK_SET_THRESHOLD

Sets the value of the threshold used in pick search operations in the current segment.

FORMAT

```
GM_$PICK_SET_THRESHOLD (threshold, status)
```

INPUT PARAMETERS

threshold

The pick threshold value. This is a 4-byte integer.

The pick threshold is initialized to 1 (all segments with nonzero pickable values are pickable).

OUTPUT PARAMETERS

status

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

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

Use GM_$PICK_INQ_THRESHOLD to obtain the current value of the pick threshold.

GM_$PICK_TRANSFORM_POINT

Transforms the coordinates of a point from the coordinate system of the viewport segment to the coordinate system of the picked segment.

FORMAT

```
GM_$PICK_TRANSFORM_POINT (vsegment_position, psegment_position, status)
```

INPUT PARAMETERS

vsegment_position
A point in viewport coordinates, in GM_$POINTREAL_T format. See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

psegment_position
A point in picked segment coordinates in GM_$POINTREAL_T format. See the GM_$ Data Types section for more information.

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

USAGE

Strictly speaking, in this context "picked segment" means a selected instance of a segment.

GM_$PLANE_MASK

Inserts a command into the current segment: change the plane mask.

FORMAT

```
GM_$PLANE_MASK (mask, status)
```

INPUT PARAMETERS

mask

The plane mask, specifying which planes to use, in GM_$PLANE_MASK_T format. This is a 2-byte integer.

The default value is [0...7], in GM_$PLANE_MASK_T format, or 255 when expressed as a 2-byte integer. The default is that all planes are in use and can be modified.

FORTRAN programmers should encode the plane mask in a 2-byte integer in the range of 0-255 (1 means plane 0 is on, 2 means plane l is on, 3 means planes 0 and 1 are on; 255 means planes 0 through 7 are on).

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_PLANE_MASK to get the value stored for the current GM_$PLANE_MASK command.

FORTRAN programmers might want to include the parameter definitions given below:

```
      integer*2
     +    bit0,
     +    bit1,
     +    bit2,
     +    bit3,
     +    bit4,
     +    bit5,
     +    bit6,
     +    bit7
      parameter (
     +    bit0  16#0001,
     +    bit1  16#0002,
     +    bit2  16#0004,
     +    bit3  16#0008,
     +    bit4  16#0010,
     +    bit5  16#0020,
     +    bit6  16#0040,
     +    bit7  16#0080)
```

```
Example:

In FORTRAN, to enable planes 2 and 5, use the following:

    CALL GM_$PLANE_MASK( bit2 + bit5, status )

In Pascal, to enable planes 2 and 5, use the following:

    GM_$PLANE_MASK( [ 2, 5 ], status )
```

GM_$POLYLINE_2D[16,32,REAL]

Inserts a command into the current segment: draw a linked set of line segments.

FORMAT

```
GM_$POLYLINE_2D16 (n_points, point_array, close, fill, status)

GM_$POLYLINE_2D32 (n_points, point_array, close, fill, status)

GM_$POLYLINE_2DREAL (n_points, point_array, close, fill, status)
```

INPUT PARAMETERS

n_points
The number of points in the list of points. This is a 2-byte integer. Polylines are limited to 1000 (GM_$MAX_ARRAY_LENGTH) points.

point_array
A list of coordinate points, each a pair (x,y) of integers in the appropriate format:

GM_$POINT16_T
A two-element array of 2-byte integers for GM_$POLYLINE_2D16

GM_$POINT32_T
A two-element array of 4-byte integers for GM_$POLYLINE_2D32

GM_$POINTREAL
A two-element array of real values for GM_$POLYLINE_2DREAL

See the GM_$ Data Types section for more information.

close
A Boolean (logical) value which specifies whether the first and last points are connected. Set the parameter to true to close the polygon. You must use close when you want to fill a polygon.

fill
A Boolean (logical) value which specifies whether to fill the polygon or not. Filled polygons must be closed. Set the parameter to true to fill the polygon; set it to false for an unfilled polygon.

OUTPUT PARAMETERS

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

USAGE

Use GM_$INQ_POLYLINE_2D[16,32,REAL] to retrieve the parameters of a polyline command inserted by GM_$POLYLINE_2D[16,32,REAL].

Currently, you must use GM_$INQ_POLYLINE_2D16 if the stored data type is GM_$16; you must use GM_$INQ_POLYLINE_2D32 or _2DREAL if the stored data type is GM_$32.

Selecting close = false and fill = true results in an error.

Before supplying coordinate data to GM_$POLYLINE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$PRIMITIVE_2D[16,32,REAL]

Inserts a command into the current segment: draw a primitive.

FORMAT

```
GM_$PRIMITIVE_2D16 (primitive_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$PRIMITIVE_2D32 (primitive_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$PRIMITIVE_2DREAL (primitive_type, n_points, point_array, n_parameters,
                      parameter_array, status)
```

INPUT PARAMETERS

primitive_type
The user-defined type of primitive command. This is a 2-byte integer.

Each distinct value of primitive_type corresponds to a different user-defined primitive display routine. For each primitive_type you use, you must write a user-defined display routine to be used when displaying (GM_$PRIMITIVE_2D) commands of that primitive type. You define a specified display routine to be used for displaying a specified primitive type using the routine GM_$PRIMITIVE_DISPLAY_2D.

n_points
The number of points in the list of points. This is a 2-byte integer. The number of points is limited to 1000 (GM_$MAX_ARRAY_LENGTH).

point_array
A list of coordinates of points each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$PRIMITIVE_2D16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$PRIMITIVE_2D32
GM_$POINTREAL	A two-element array of real values for GM_$PRIMITIVE_2DREAL

See the GM_$ Data Types section for more information.

n_parameters
The number of parameters in the list of parameters. This is a 2-byte integer.

parameter_array
A list of parameters, in GM_$ARRAYREAL_T format. This is an array of real values.

OUTPUT PARAMETERS

status

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

USAGE

Before supplying coordinate data to GM_$PRIMITIVE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$PRIMITIVE_DISPLAY_2D

Assigns the specified user-defined routine to the specified user-defined primitive type number.

FORMAT

```
GM_$PRIMITIVE_DISPLAY_2D (primitive_type, display_procedure_ptr, status)
```

INPUT PARAMETERS

primitive_type
The user-defined type of primitive command. This is a 2-byte integer.

display_procedure_ptr
Entry point for the application-supplied procedure that displays (GM_$PRIMITIVE_2D) commands of the specified primitive type, in GM_$PRIMITIVE_PTR_T format. This is a pointer to a procedure.

When a (GM_$PRIMITIVE_2D) command of the specified primitive type is encountered during display operations, the graphics metafile package calls the application-supplied procedure to display the command. Four input parameters are passed to the application-supplied procedure:

```
n_points: the number of points in point_array.  This is
          a 2-byte integer.

point_array: the list of points, transferred to display
             coordinates.  This is an array of pairs
             (x,y) of 2-byte integers.

n_parameters: the number of parameters in parameter_array.
              This is a 2-byte interger.

parameter_array: the list of parameters.  This is an
                 array of reals.
```

If you use a value of NIL for display_procedure_ptr, no routine is called at display time. You can use this to undo an assignment of a procedure to a specified user-defined primitive-type number.

In FORTRAN, pass procedure pointers as indicated in the description of GM_$REFRESH_SET_ENTRY. Use 0 (not NIL) to indicate a zero value.

OUTPUT PARAMETERS

status

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

USAGE

Successive calls to GM_$PRIMITIVE_DISPLAY_2D for the same primitive type override previously defined entry points.

The user-supplied routine may contain only GPR drawing routine calls.

GM_$PRINT_FILE

Converts the current metafile to the specified file for subsequent printing on a hard-copy device.

FORMAT

```
GM_$PRINT_FILE (file_name, file_name_length, size, invert, print_style,
                bpi, status)
```

INPUT PARAMETERS

file_name
Pathname of the output file, in NAME_$PNAME_T format. This is an array of up to 256 characters.

If you specify a file name that already exists, the old contents of the file are overwritten.

file_name_length
Number of characters in the pathname. This is a 2-byte integer.

size
Pair (x, y) of coordinates, in GM_$POINT16_T. This is a two-element array of 2-byte integers. See the GM Data Types section for more information.

invert
Boolean (logical) value specifying whether to invert the file or not. Set to true to invert the file. Set to false to print the file without inverting it.

print_style
Type of output file to be created, in GM_$PRINT_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$GMF	Specifies that you want to copy a metafile to a bitmap for storage in a graphics map file (GMF).
GM_$OUT1	Specifies that you want to create a vector command file. Only one plane, plane 0, is stored.
GM_$POSTSCRIPT	Specifies that you want to create a PostScript file

For print_style = GM_$OUT1, all coordinates are transformed to display coordinates in accordance with the size parameter of the GM_$PRINT_... routine that you used to create the vector command file. In the GM_$OUT1 file, the origin of coordinates is the top left, not the bottom left as in the metafile. The GM_$OUT1 file is scaled to the size parameter using the standard 95% rule that one dimension fills 95% of the size block, and the other dimension does not overflow the block.

For print_style = GM_$GMF, the bpi value sets the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the specified number of bits per inch. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image.

bpi

Number of bits per inch in the output GMF (graphics map file). This is a 2-byte integer.

OUTPUT PARAMETERS

status

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

USAGE

GM_$PRINT_FILE prints the primary segment of the file. To print some other segment, you may make the desired segment the primary segment, call GM_$PRINT_FILE, and then restore the previous value of the primary segment.

GM_$PRINT_FILE_PART

Converts part of the current metafile to the specified file for subsequent printing on a hard-copy device.

FORMAT

```
GM_$PRINT_FILE_PART (bounds, file_name, file_name_length, size, invert,
                     print_style, bpi, status)
```

INPUT PARAMETERS

bounds
Part of this file to be printed, in segment coordinates of the primary segment, in GM_$BOUNDSREAL_T format. This is a four-element array of real values (xmin, ymin, xmax, ymax).

file_name
Pathname of the output file, in NAME_$PNAME_T format. This is an array of up to 256 characters.

file_name_length
Number of characters in the pathname. This is a 2-byte integer.

If you specify a file name that already exists, the old contents of the file are overwritten.

size
Pair (x,y) of coordinates, in GM_$POINT16_T. This is a two-element array of 2-byte integers.

invert
Boolean (logical) value specifying whether to invert the file or not. Set to true to invert the file. Set to false to print the file without inverting it.

print_style
Type of output file to be created, in GM_$PRINT_STYLE_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$GMF	Specifies that you want to copy a metafile to a bitmap for storage in a graphics map file (GMF).
GM_$OUT1	Specifies that you want to create a vector command file.
GM_$POSTSCRIPT	Specifies that you want to create a PostScript file.

For print_style = GM_$OUT1, all coordinates are transformed to display coordinates in accordance with the size parameter of the GM_$PRINT_... routine that you used to create the vector command file. In the GM_$OUT1 file, the origin of coordinates is the top left, not the bottom left as in the metafile. The GM_$OUT1 file is scaled to the size parameter using the standard 95% rule that one dimension fills 95% of the size block, and the other dimension does not overflow the block.

For print_style = GM_$GMF, the bpi value sets the physical density of the image represented in the GMF. If this parameter is nonzero, a device to which you output the GMF may compress or expand the image to produce a result which is as close as possible to the specified number of bits per inch. If this parameter is zero, an output device uses one dot to represent each bit from the GMF, regardless of the resulting physical size of the image.

bpi
Number of bits per inch in the output GMF (graphics map file). This is a 2-byte integer.

OUTPUT PARAMETERS

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

USAGE

GM_$PRINT_FILE_PART prints the primary segment of the file. To print some other segment, you may make the desired segment the primary segment, call GM_$PRINT_FILE_PART, and then restore the previous value of the primary segment.

GM_$RECTANGLE_[16,32,REAL]

Inserts a command into the current segment : draw a rectangle with sides parallel to the x and y axes.

FORMAT

```
GM_$RECTANGLE_16 (point1, point2, fill, status)

GM_$RECTANGLE_32 (point1, point2, fill, status)

GM_$RECTANGLE_REAL (point1, point2, fill, status)
```

INPUT PARAMETERS

point1, point2

The coordinates of two diagonally opposite corners, each a pair (x,y) of values in the appropriate format:

GM_$POINT16_T	A two-element array of 2-byte integers for GM_$RECTANGLE_16
GM_$POINT32_T	A two-element array of 4-byte integers for GM_$RECTANGLE_32
GM_$POINTREAL	A two-element array of real values for GM_$RECTANGLE_REAL

fill

A Boolean (logical) value which specifies whether to fill the rectangle or not. Set the parameter to true to fill the rectangle; set it to false for an unfilled rectangle.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_RECTANGLE_[16,32,REAL] to retrieve the parameters of a rectangle command inserted by GM_$RECTANGLE_[16,32,REAL].

Currently, you must use GM_$INQ_RECTANGLE_16 if the stored data type is GM_$16; you must use GM_$INQ_RECTANGLE_32 or _REAL if the stored data type is GM_$32.

Before supplying coordinate data to GM_$RECTANGLE_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$REFRESH_SET_ENTRY

Specifies a user-defined routine to be called when the display is refreshed as a result of a DM refresh window or POP command.

FORMAT

```
GM_$REFRESH_SET_ENTRY (refresh_procedure_ptr, status)
```

INPUT PARAMETERS

refresh_procedure_ptr

refresh_procedure_ptr
Entry point for the application-supplied procedure to refresh the display, in GM_$REFRESH_PTR_T format. This is a pointer to procedure.

In direct mode, when the Display Manager refreshes the window in which the GM bitmap is contained, the specified application-supplied procedure is called. Two input parameters are passed to the application-supplied procedure:

```
unobscured    When false, this Boolean value indicates
              that the window is obscured.

position_changed
              When true, this Boolean value indicates
              that the window has moved or grown since
              the display was released.
```

The "pointer to procedure" data type is an extension to the Pascal language. See the *DOMAIN Pascal Language Reference* for an explanation of this extension.

This routine requires you to pass procedure pointers. To do so in FORTRAN programs, use the following technique. First declare the subroutines to be passed as EXTERNAL. Then pass their names using the IADDR function to simulate the Pascal pointer mechanism. For example:

```
EXTERNAL REFRESH_WINDOW
.
.
.
CALL GPR_$SET_REFRESH_ENTRY (IADDR(REFRESH_WINDOW),IADDR,
STATUS)
```

In FORTRAN, use 0 (not NIL) to indicate a zero value.

OUTPUT PARAMETERS

status

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

USAGE

Successive calls to GM_$REFRESH_SET_ENTRY override previously defined entry points.

In within-GPR mode, use GPR_$SET_REFRESH_ENTRY.

GM_$REPLACE_INQ_FLAG

Returns the current value of the replace flag (Obsolete). New programs use GM_$MODELCMD_INQ_MODE.

FORMAT

```
GM_$REPLACE_INQ_FLAG (yes_no, status)
```

OUTPUT PARAMETERS

yes_no
A Boolean value indicating whether the replace flag is set. True indicates that the flag is set (new commands replace the current command); false indicates that the flag is cleared (new commands are inserted after the current command). The default value is false.

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

USAGE

This routine is functional, but new programs should use GM_$MODELCMD_INQ_MODE.

GM_$REPLACE_SET_FLAG

Sets or clears a flag which causes subsequent commands to replace the current command rather than being inserted after it (Obsolete). New programs use GM_$MODELCMD_SET_MODE.

FORMAT

```
GM_$REPLACE_SET_FLAG (yes_no, status)
```

INPUT PARAMETERS

yes_no

A Boolean value indicating whether the replace flag is set. Use true to set the flag (new commands replace the current command); use false to clear the flag (new commands are inserted after the current command).

The default value is false (new commands are inserted after the current command).

When the replace flag is set, and you call a routine which creates a command of the same command type as the current command, the new command replaces the current command.

If you call a routine which creates a different command type, the replace flag is automatically cleared and the new command is inserted after the current command.

Changing the current command (for example, by calling GM_$COMMAND_DELETE) automatically clears the replace flag.

OUTPUT PARAMETERS

status

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

USAGE

This routine is functional, but new programs should use GM_$MODELCMD_SET_MODE.

Only primitive and instance command types may be replaced. The replace flag may only be set if the current command is a primitive or instance command.

GM_$SEGMENT_CLOSE

Closes the current segment, saving revisions or not.

FORMAT

```
GM_$SEGMENT_CLOSE (save, status)
```

INPUT PARAMETERS

save

A Boolean (logical) value that indicates whether to save revisions. Set to true to save revisions; set to false not to save revisions.

You must set save to true. Do not assume that it is true by default.

OUTPUT PARAMETERS

status

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

GM_$SEGMENT_COPY

Copies the entire contents of another segment into the current segment.

FORMAT

```
GM_$SEGMENT_COPY (segment_id, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to be copied, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

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

USAGE

The entire contents of the specified segment are copied into the current segment after the current command. The current command is set equal to the last copied command.

You cannot copy a segment from one file to another file.

Use the following procedure to make a new segment named "newcopy," which is an exact copy of an existing segment. The identification of the existing segment is 'source_seg_id':

```
GM_$SEGMENT_CREATE ('newcopy', 7, segment_id, status);
GM_$SEGMENT_COPY (source_seg_id, status)
GM_$SEGMENT_CLOSE (true, status)
```

The two copies may then be edited independently and instanced independently.

GM_$SEGMENT_CREATE

Creates a new segment.

FORMAT

```
GM_$SEGMENT_CREATE (name, name_length, segment_id, status)
```

INPUT PARAMETERS

name

The pathname of the segment, in NAME_$PNAME_T format. This is a character string.

Currently, the segment name is truncated to twelve characters.

Segments in the same file must have different segment names. Note that "SEG" and "seg" are different segment names; the comparison is case-sensitive.

Verification that each name is unique carries a performance penalty. Therefore, you have the option of not naming segments and using the the segment identification number to reference segments. To create an unnamed segment, set the value for name to 0:

```
GM_$SEGMENT_CREATE ( '', 0, seg_id, status)
```

You can use GM_$SEGMENT_RENAME to give a name to an unnamed segment or to remove the name of a segment.

name_length

The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

segment_id

The identification number assigned to the segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

status

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

USAGE

The segment name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another

You must close the current segment before creating a new segment.

When a segment is created, its pickable and visible values are set to 255.

For a segment name, you can use any collection of byte values of length 1 through 12. Trailing blanks in segment names are not discarded.

If you are careful, you may use a number for the segment name:

```
VAR
     number: integer32;
 .
 .
 .
GM_$SEGMENT_CREATE(number,4,seg_id,status);
```

GM_$SEGMENT_DELETE

Deletes the current segment.

FORMAT

```
GM_$SEGMENT_DELETE (status)
```

OUTPUT PARAMETERS

status

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

USAGE

There must be no references to the deleted segment.

If you delete a segment, its identification number will be reassigned. The smallest unused identification number is reassigned first.

You may not delete the file's primary segment. If you attempt to do so, you will get this error message: gm_$illegal_value.

GM_$SEGMENT_ERASE

Deletes all commands in the current segment.

FORMAT

```
GM_$SEGMENT_ERASE (status)
```

OUTPUT PARAMETERS

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

USAGE

After this routine is performed, the current segment is still the current segment, but it contains no commands.

GM_$SEGMENT_INQ_BOUNDS

Returns the bounds of a segment.

FORMAT

```
GM_$SEGMENT_INQ_BOUNDS (bounds,status)
```

INPUT PARAMETERS

seg_id
Segment ID in GM_$SEGMENT_ID_T format. This is a positive 4-byte integer.

OUTPUT PARAMETERS

bounds
Bounds of the segment in GM_$BOUNDSREAL_T format. This is a four-element array of real numbers. See the GM Data Types section for more information.

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

USAGE

Use this call to obtain the coordinates of the bottom left-hand boundary and the top right-hand boundary of the segment.

Use GM_$FILE_INQ_BOUNDS to obtain the bounds of the primary segment of a file.

Use GM_$COMMAND_INQ_BOUNDS to obtain the bounds of the current command.

GM_$SEGMENT_INQ_COUNT

Returns the number of segments in this metafile and a segment number guaranteed to be greater than or equal to the largest segment number.

FORMAT

```
GM_$SEGMENT_INQ_COUNT (count, max_segid, status)
```

OUTPUT PARAMETERS

count
The number of segments in the metafile, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

max_segid
A number greater than or equal to the largest segment ID in the file, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

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

USAGE

When you retrieve the count and maximum segment ID, you can then look at every segment by checking segment numbers from 0 to this maximum value (0 is used).

GM_$SEGMENT_INQ_CURRENT

Returns the name, segment identification, and number of instances of the current segment.

FORMAT

```
GM_$SEGMENT_INQ_CURRENT (name, name_length, segment_id,
                         n_instances, status)
```

OUTPUT PARAMETERS

name
The pathname of the segment, in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

segment_id
The identification number assigned to the segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

n_instances
The number of instances of the segment. This is a 4-byte integer.

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

USAGE

The returned segment number must be used for creating references to this segment within other segments.

GM_$SEGMENT_INQ_ID

Returns the segment identification and the number of instances of the named segment.

FORMAT

```
GM_$SEGMENT_INQ_ID (name, name_length, segment_id, n_instances, status)
```

INPUT PARAMETERS

name
The pathname of the segment. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

OUTPUT PARAMETERS

segment_id
The identification number assigned to the segment of specified name, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

In creating instances of (references to) this segment within other segments, you must use the returned segment identification number.

n_instances
The number of instances of the segment. This is a 4-byte integer.

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

USAGE

The name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another.

GM_$SEGMENT_INQ_ID is complementary to GM_$SEGMENT_INQ_NAME.

Only the current file is searched to identify the segment number and the number of instances of the named segment.

GM_$SEGMENT_INQ_NAME

Returns the name of the segment with the specified segment identification number.

FORMAT

```
GM_$SEGMENT_INQ_NAME (seg_id, name, name_length, n_instances, status)
```

INPUT PARAMETERS

segment_id
The identification number assigned to the segment, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

name
The pathname of the segment, in NAME_$PNAME_T format. This is an array of up to 256 characters.

name_length
The number of characters in the pathname. This is a 2-byte integer.

n_instances
The number of instances of the segment. This is a 4-byte integer.

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

USAGE

The name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another.

GM_$SEGMENT_INQ_NAME is complementary to GM_$SEGMENT_INQ_ID.

GM_$SEGMENT_INQ_PICKABLE

Returns the pickable value of the specified segment.

FORMAT

```
GM_$SEGMENT_INQ_PICKABLE (segment_id, pickable, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the pickable value is to be retrieved, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

pickable
The pickable value of the specified segment. This is a 4-byte integer.

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

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$SEGMENT_INQ_TEMPORARY

Returns whether the specified segment is temporary or not.

FORMAT

```
GM_$SEGMENT_INQ_TEMPORARY (segment_id, temporary, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the temporary/permanent status is to be retrieved, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

temporary
A Boolean (logical) value that indicates whether the segment is temporary. A value of true indicates that the segment is temporary; false indicates that the segment is permanent.

Temporary segments are deleted when the file is closed.

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

GM_$SEGMENT_INQ_VISIBLE

Returns the visible value of the specified segment.

FORMAT

```
GM_$SEGMENT_INQ_VISIBLE (segment_id, visible, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the visible value is to be retrieved, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

visible
The visible value of the specified segment. This is a 4-byte integer.

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

GM_$SEGMENT_OPEN

Reopens an existing segment.

FORMAT

```
GM_$SEGMENT_OPEN (segment_id, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to open, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

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

USAGE

You must close the current segment before opening another segment.

Use GM_$SEGMENT_INQ_CURRENT to get the identification number of the current segment.

Currently, you cannot open a segment in a file that is open for read access. To open a segment, the file containing the segment must be open in write or read-write access mode. Otherwise, this error message is returned: gm_$illegal_value.

GM_$SEGMENT_RENAME

Renames an existing segment.

FORMAT

```
GM_$SEGMENT_RENAME (segment_id, name, name_length, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment to rename, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

The segment number remains the same when you rename the segment.

name
The new name of the segment in NAME_$PNAME_T format. This is an array of up to 256 characters.

If another segment already has the new name, you receive an error message, and the old name is not changed.

name_length
The number of characters in the new name of the segment. This is a 2-byte integer.

Currently, the segment name is truncated to twelve characters.

OUTPUT PARAMETERS

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

USAGE

Verification that each name is unique carries a performance penalty. Therefore, you have the option of not naming segments and using the segment identification number to reference segments. To create an unnamed segment, set the value for name to 0:

```
GM_$SEGMENT_CREATE ( '', 0, seg_id, status)
```

You can use GM_$SEGMENT_RENAME to give a name to an unnamed segment or to remove the name of a segment.

The name is of arbitrary length; however, currently only the first twelve characters are stored to differentiate one segment from another.

To find the segment_id of an existing segment for which you know the name, use GM_$SEGMENT_INQ_ID.

GM_$SEGMENT_SET_PICKABLE

Assigns a pickable value to the specified segment.

FORMAT

```
GM_$SEGMENT_SET_PICKABLE (segment_id, pickable, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the pickable value is to be changed, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

pickable
The pickable value for the specified segment. This is a 4-byte integer.

When a segment is created, its pickable value is initialized to 255.

OUTPUT PARAMETERS

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

USAGE

The pick mask and threshold values are used during pick segment operations to determine which segments are pickable. The segment is pickable if two conditions are met: the segment's pickable value is greater than or equal to the pick threshold; and the result of a bitwise-AND of the segment pickable value and the pick mask is nonzero. If either condition is not met, the segment is not picked.

GM_$SEGMENT_SET_TEMPORARY

Makes the specified segment temporary or not. Temporary segments are deleted when the file is closed.

FORMAT

```
GM_$SEGMENT_SET_TEMPORARY (segment_id, temporary, status)
```

INPUT PARAMETERS

segment_id

The identification number of the segment to make temporary, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

temporary

A Boolean value that indicates whether the segment is temporary. Set to true to make temporary; set to false to make permanent.

When a segment is created, it is made permanent (temporary = false).

A temporary segment is useful for picture data that you want to display now but not store for future use.

OUTPUT PARAMETERS

status

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

GM_$SEGMENT_SET_VISIBLE

Assigns a visible value to the specified segment.

FORMAT

```
GM_$SEGMENT_SET_VISIBLE (segment_id, visible, status)
```

INPUT PARAMETERS

segment_id
The identification number of the segment for which the visible value is to be changed, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

visible
The visible value for the specified segment. This is a 4-byte integer.

When a segment is created, its visible value is initialized to 255.

OUTPUT PARAMETERS

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

USAGE

GM_$SEGMENT_SET_VISIBLE lets you display a picture without certain segments.

GM_$TAG

Inserts a comment into the current segment.

FORMAT

```
GM_$TAG (string, string_length, status)
```

INPUT PARAMETERS

string
The text string to write, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length
The length of the string. This is a 2-byte integer.

OUTPUT PARAMETERS

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

USAGE

Use GM_$INQ_TAG to get the value stored for the current GM_$TAG command.

Descriptor tags in a stroke font file must be entered in the file in capital letters, for example, WIDTH.

GM_$TAG_LOCATE

Looks for the specified tag in the specified range of segments and returns the segment ID of the lowest numbered segment in which the tag is found.

FORMAT

```
GM_$TAG_LOCATE (string, string_length, min, max, segment_id, status)
```

INPUT PARAMETERS

string

The string to be searched for, in GM_$STRING_T format. This is an array of up to 120 characters.

The string to be matched is passed through the pathname wildcard parser, as described in *DOMAIN System Command Reference* manual. To guarantee noninterference with the wildcard parser, you may place an escape character (@) between every byte of the string you wish to search for.

string_length

The length of the string to be searched for. This is a 2-byte integer.

min

The smallest segment number to search, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

max

The largest segment number to search, in GM_$SEGMENT_ID_T format. This is a 4-byte integer.

OUTPUT PARAMETERS

segment_id

The number of the segment in which the tag was found, in GM_$SEGMENT_ID_T format. This is 4-byte integer.

status

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

USAGE

To find all occurrences of a tag, you must make successive calls to GM_$TAG_LOCATE.

GM_$TERMINATE

Terminates the graphics metafile package and closes the display.

FORMAT

```
GM_$TERMINATE (status)
```

OUTPUT PARAMETERS

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

USAGE

When GM terminates, it currently resets color 1 to whatever it was when GM was initialized. This is true of color nodes only. If you use GM in borrow mode, the entire color map is reset when GM terminates. (The resetting is not by GM, but by GPR.)

Any open files are closed. Revisions to these files are saved.

Any open segments are closed, and revisions are saved.

GM_$TEXT_2D[16,32,REAL]

Inserts a command into the current segment: write a text string.

FORMAT

```
GM_$TEXT_2D[16,32,REAL] (point, rotate, string, string_length, status)
```

INPUT PARAMETERS

point

The coordinates of the point at which to locate text. This is a pair (x,y) of values in the appropriate format:

GM_$POINT16_T
: A two-element array of 2-byte integers for GM_$TEXT_2D16

GM_$POINT32_T
: A two-element array of 4-byte integers for GM_$TEXT_2D32

GM_$POINTREAL
: A two-element array of real values for GM_$TEXT_2DREAL

See the GM_$ Data Types section for more information.

The text is placed as follows: The first character of the text string is placed at the location you specify. This means that the origin of this character, as defined in the font, is placed at the specified location. Usually, the origin is the lower left-hand corner, excluding descenders.

rotate

The angle at which this text string is to be written, in degrees. This is a real variable. Use 0.0 degrees to specify left to right text. Other values indicate clockwise rotation. For example, -90.0 degrees specifies bottom to top.

string

The text string to write, in GM_$STRING_T format. This is an array of up to 120 characters.

string_length

The length of the string. This is a 2-byte integer.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_TEXT_2D[16,32,REAL] to retrieve the parameters of a text command inserted by GM_$TEXT_2D[16,32,REAL].

Before supplying coordinate data to GM_$TEXT_2DREAL, you must call GM_$DATA_COERCE_SET_REAL. This forces real variables that you send to the package to be stored in 32-bit storage format.

GM_$TEXT_BACKGROUND_VALUE

Inserts a command into the current segment: change the background value used when writing text.

FORMAT

```
GM_$TEXT_BACKGROUND_VALUE (value, status)
```

INPUT PARAMETERS

value

The value to use for the text background. This is a 4-byte integer.

The default value is -2. This sets the text background value equal to the viewport background value.

The value -1 makes the background transparent: the text background value is equal to the current display pixel value.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_TEXT_BACKGROUND_VALUE to get the value stored for the current GM_$TEXT_BACKGROUND_VALUE command.

GM_$TEXT_SIZE

Inserts a command into the current segment : use a different text size from the same font family.

FORMAT

```
GM_$TEXT_SIZE (size, status)
```

INPUT PARAMETERS

size

The maximum character height, in segment coordinates of the viewport primary segment, which may be used to display text. This is a real value.

The default text size is 10.0.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_TEXT_SIZE to retrieve the parameters of a text command inserted by GM_$TEXT_SIZE.

GM_$TEXT_VALUE

Inserts a command into the current segment: set the value used when writing text.

FORMAT

```
GM_$TEXT_VALUE (value, status)
```

INPUT PARAMETERS

value

The value that specifies the new value to use when writing text. This is a 4-byte integer.

The default value is 1.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$INQ_TEXT_VALUE to get the value stored for the current GM_$TEXT_VALUE command.

GM_$VIEW_SCALE

Scales the view under the current viewport, keeping the specified point fixed.

FORMAT

```
GM_$VIEW_SCALE (scale, point, status)
```

INPUT PARAMETERS

scale

The value by which to multiply the view scale factor. This is a real value.

point

An (x,y) pair indicating the fixed point on screen, in GM_POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

The point (point.x,point.y) on the screen (expressed in fraction-of-bitmap coordinates) is kept fixed during this rescaling.

OUTPUT PARAMETERS

status

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

USAGE

Use the following to rescale the screen by a scaling factor "scale" AND move the point (point.x,point.y) on the screen (in fraction-of-bitmap coordinates) to the center of the viewport, all in one operation:

```
{assumes scale not equal to 1.0}

GM_$VIEWPORT_INQ_BOUNDS (vbounds, status);
vcenter_x := 0.5 * (vbounds.xmax + vbounds.xmin);
vcenter_y := 0.5 * (vbounds.ymax + vbounds.ymin);
point1.x := (vcenter_x - point.x * scale)/(1.0 - scale);
point1.y := (vcenter_y - point.y * scale)/(1.0 - scale);
GM_$VIEW_SCALE(scale, point1,status);
```

GM_$VIEW_TRANSFORM

Rotates the view under the current viewport, keeping the specified point (in fraction-of-bitmap coordinates) fixed.

FORMAT

```
GM_$VIEW_TRANSFORM (rotate, point, status)
```

INPUT PARAMETERS

rotate

The rotation to be applied to coordinates in the segment, in GM_$ROTATE_REAL2x2_T format. This is a four-element array of real values (xx,xy,yx,yy), where the second element (xy) represents the dependence of the x-result on the y-source. See the GM_$ Data Types section for more information.

point

An (x,y) pair indicating the fixed point on the screen, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

The point (point.x,point.y) on the screen (expressed in fraction-of-bitmap coordinates) is kept fixed during this transformation.

OUTPUT PARAMETERS

status

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

GM_$VIEW_TRANSFORM_RESET

Resets the view tranformation to the form in which it was initially displayed.

FORMAT

```
GM_$VIEW_TRANSFORM_RESET (status)
```

OUTPUT PARAMETERS

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

USAGE

GM_$VIEW_TRANSFORM_RESET undoes the effects of these commands: GM_$VIEW_TRANSLATE, GM_$VIEW_SCALE, and GM_$GM_VIEW_TRANSFORM. However, GM_$VIEW_TRANSFORM_RESET does not undo changes caused by changing the window size.

GM_$VIEW_TRANSLATE

Translates the view under the current viewport.

FORMAT

```
GM_$VIEW_TRANSLATE (translate, status)
```

INPUT PARAMETERS

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values. See the GM_$ Data Types section for more information.

The translation is specified in bitmap coordinates, that is, as fractions of the display bitmap.

A positive x translation moves the viewport to the right over the view, so that the picture on the display appears to move to the left. A positive y translation moves the viewport up over the view, so that the picture on the display appears to move down.

OUTPUT PARAMETERS

status

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

GM_$VIEWPORT_CLEAR

Clears the current viewport.

FORMAT

```
GM_$VIEWPORT_CLEAR (value, status)
```

INPUT PARAMETERS

value
The value to which all pixels within the current viewport are to be set. This is a 4-byte integer.

OUTPUT PARAMETERS

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

USAGE

Only planes enabled by the current value of the plane mask are affected.

GM_$VIEWPORT_CREATE

Creates an additional viewport and makes it the current viewport.

FORMAT

```
GM_$VIEWPORT_CREATE (bounds, viewport_id, status)
```

INPUT PARAMETERS

bounds
The bounds of the new viewport, in GM_$BOUNDSREAL_T format. This is an array of four real values (xmin, ymin, xmin, ymax). See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is assigned by the GM package.

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

USAGE

GM_$INIT initializes the GM package and viewports, creates one viewport called viewport 1 which fills the display bitmap and makes it the selected viewport. Currently, viewports may not overlap, so you must change the bounds of viewport 1 before creating additional viewports. You must supply bounds for the new viewport, in bitmap coordinates. The GM package assigns a number to the viewport.

Use this procedure to change the original viewport to fill only the left half of the screen and create a second viewport in the center right of the screen:

```
bounds.xmin := 0.0; bounds.ymin := 0.0;
bounds.xmax := 0.5; bounds.ymax := 1.0;
GM_$VIEWPORT_SET_BOUNDS (bounds, status);
bounds.xmin := 0.6; bounds.ymin := 0.25;
bounds.xmax := 1.0; bounds.ymax := 0.75;
GM_$VIEWPORT_CREATE (bounds, viewport_id, status);
```

GM_$VIEWPORT_DELETE

Deletes a viewport.

FORMAT

```
GM_$VIEWPORT_DELETE (viewport_id, status)
```

INPUT PARAMETERS

viewport_id
The number assigned by the GM package to the viewport you wish to delete. This is a 2-byte integer.

OUTPUT PARAMETERS

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

USAGE

Because viewports currently may not overlap, you must delete all but one viewport if a single viewport is to be expanded to fill the entire GM bitmap.

GM_$VIEWPORT_INQ_BACKGROUND_VALUE

Returns the pixel value used for the background of the specified viewport.

FORMAT

```
GM_$VIEWPORT_INQ_BACKGROUND_VALUE (viewport_id, value, status)
```

INPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is the one assigned by the GM package.

OUTPUT PARAMETERS

value
The value to use for the viewport background. This is a 4-byte integer.

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$VIEWPORT_INQ_BACKGROUND_VALU | E
```

Use GM_$VIEWPORT_SET_BACKGROUND_VALUE to change the background value of the specified viewport.

GM_$VIEWPORT_INQ_BORDER_SIZE

Returns the border size of the current viewport, in pixels or fraction-of-bitmap coordinates.

FORMAT

```
GM_$VIEWPORT_INQ_BORDER_SIZE (border_unit, border_size, status)
```

OUTPUT PARAMETERS

border_unit
The units for border size, in GM_$BORDER_UNIT_T format. This is a 2-byte integer. One of the following values is returned:

GM_$FRACTIONS	Expresses edge width as fraction of the total 6M bitmap size.
GM_$PIXELS	Default border type. Expresses edge width in pixels.

border_size
The size of the border, specified as left, bottom, right, top. This is an array of four real values (left, bottom, right, top).

The default border type is in pixels, and the default width is 1,1,1,1.

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

USAGE

GM_$VIEWPORT_INQ_BORDER_SIZE returns the size of the four edges of the current viewport. If border_unit = GM_$PIXELS, edge widths are expressed in pixels. If border_unit = GM_$FRACTIONS, edge widths are expressed as fractions of the total GM bitmap size.

Use GM_$VIEWPORT_SET_BORDER_SIZE to change the size of the border.

GM_$VIEWPORT_INQ_BOUNDS

Returns the bounds of the current viewport.

FORMAT

```
GM_$VIEWPORT_INQ_BOUNDS (bounds, status)
```

OUTPUT PARAMETERS

bounds

The bounds of the current viewport, in GM_$BOUNDSREAL_T format. This is a four element array of real values (xmin, ymin, xmax, ymax). See the GM_$ Data Types section for more information.

status

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

USAGE

GM_$VIEWPORT_INQ_BOUNDS returns the bounds of the current viewport, as fractions of the total GM bitmap size.

GM_$VIEWPORT_INQ_CURRENT

Returns the number of the current viewport.

FORMAT

```
GM_$VIEWPORT_INQ_CURRENT (viewport_id, status)
```

OUTPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is assigned by the GM package.

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

USAGE

If there is no current viewport, a GM_$NO_CURRENT_VIEWPORT error is returned.

GM_$VIEWPORT_INQ_GRIDS

Returns the number and types of grids for a viewport.

FORMAT

```
GM_$VIEWPORT_INQ_GRIDS (maxcnt, flags, sindex, cnt, grid, status)
```

INPUT PARAMETERS

maxcnt
Length of the grid array. This is a 2-byte integer.

OUTPUT PARAMETERS

flags
Attributes of the snap grid, in GM_$GRID_FLAGS_T format. This is a 2-byte integer. Snapping is not currently implemented.

sindex
Index of the snap grid. This is a 2-byte integer. Snapping is not currently implemented.

cnt
Number of grids. This is a 2-byte integer.

grid
Array of grid descriptions, in GM_$GRID_ARRAY_T format. This is an array of [1 .. gm_$max_grid] of GM_$GRID_T. See the GM Data Types section for more information.

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

USAGE

Snapping has not been implemented for grids.

Use GM_$VIEWPORT_SET_GRIDS to establish a grid for a viewport.

GM_$VIEWPORT_INQ_REFRESH_STATE

Returns the refresh state of the current viewport.

FORMAT

```
GM_$VIEWPORT_INQ_REFRESH_STATE (refresh_state, status)
```

OUTPUT PARAMETERS

refresh_state

The refresh state of the viewport, in GM_$VIEW_REFRESH_T format. This is a 2-byte integer. One of the following values is returned:

GM_$REFRESH_INHIBIT
: In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH, or when the display is refreshed as the result of a DM command which causes the window to be redrawn. Thus, calling GM_$DISPLAY_REFRESH does not affect a viewport in this refresh state.

GM_$REFRESH_WAIT
: (Default) In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH or when the display is refreshed as the result of a DM command which causes the window to be redrawn.

GM_$REFRESH_PARTIAL
: Every time you change any command in the file, the following occurs if this viewport is the current viewport: Inserted primitive commands are added, and deleted primitive commands are erased, but underlying data is not rewritten. This provides faster interactive drawing. You should, however, periodically clean up the accumulating inaccuracies by calling GM_$VIEWPORT_REFRESH to redisplay the viewport.

GM_$REFRESH_UPDATE
: Every time you change any command in the file, this viewport is completely corrected if it is the current viewport.

status

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

USAGE

The viewport refresh states are defined under the routine GM_$VIEWPORT_SET_REFRESH_STATE.

GM_$VIEWPORT_MOVE

Translates the current viewport, carrying the view with it.

FORMAT

```
GM_$VIEWPORT_MOVE (translate, status)
```

INPUT PARAMETERS

translate

An (x,y) pair indicating the amount of translation, in GM_$POINTREAL_T format. This is a two-element array of real values.

The translation is expressed as fractions of the display bitmap size.

Currently, values which would cause part of the viewport to be moved outside the GM bitmap result in an error.

OUTPUT PARAMETERS

status

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

GM_$VIEWPORT_PIXEL_TO_SEG_2D

Converts pixel coordinates to segment coordinates.

FORMAT

```
GM_$VIEWPORT_PIXEL_TO_SEG_2D (pixel_position, viewport_id,
                              segment_position, status)
```

INPUT PARAMETERS

pixel_position
Pixel coordinates of any point on the screen in GM_$POINT16_T format. This is a two-element array of 2-byte integers.

OUTPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer.

segment_position
Segment coordinates of the point in the viewport segment of the point whose coordinates were entered in segment_position. This parameter uses GM_$POINTREAL_T format, which is a two-element array of real values.

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

USAGE

This routine allows the user to inquire about the segment position of a particular point on the screen.
The GM_$VIEWPORT_PIXEL_TO_SEG_2D call can only be used in GM_$CURRENT_BITMAP mode. Use this call when using 2D GMR with DOMAIN/Dialogue.

GM_$VIEWPORT_REFRESH

Refreshes the current viewport.

FORMAT

```
GM_$VIEWPORT_REFRESH (status)
```

OUTPUT PARAMETERS

status

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

GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL

Converts segment coordinates to pixel coordinates.

FORMAT

```
GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL (segment_position, pixel_position,
                                          status)
```

INPUT PARAMETERS

segment_position
Segment coordinates of any point in the viewport segment, in GM_$POINT16_T format. This is a two-element array of 2-byte integers, 4-byte integers, or real numbers depending on which form of the call is used.

OUTPUT PARAMETERS

pixel_position
Pixel coordinates of the screen location whose coordinates were entered in parameter segment_position. This parameter uses GM_$POINT16_T format. This is a two-element array of 2-byte integers, 4-byte integers, or real numbers depending on which form of the call is used.

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

USAGE

This routine allows the user to inquire about the pixel location of a particular point in a segment. For example, if a rectangle is drawn from (0,0) to (100,100), the user can obtain the screen location (pixel coordinates) of the point (0,0), or any other point in the segment.

This call is useful when drawing grids. For example, consider drawing the lines of a grid every 100 segment coordinates. You can use this call to determine where the points (0,0), and (100,0) will be drawn. This allows you to determine the density of your grid.

GM_$VIEWPORT_SELECT

Makes a viewport the current viewport.

FORMAT

```
GM_$VIEWPORT_SELECT (viewport_id, status)
```

INPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is the one assigned by the GM package.

OUTPUT PARAMETERS

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

USAGE

You must create the viewport before you can select it.

GM_$VIEWPORT_SET_BACKGROUND_VALUE

Sets the pixel value used for the background of the specified viewport.

FORMAT

```
GM_$VIEWPORT_SET_BACKGROUND_VALUE (viewport_id, value, status)
```

INPUT PARAMETERS

viewport_id
The number of the viewport. This is a 2-byte integer. The number is the one assigned by the GM package.

value
The value to use for the viewport background. This is a 4-byte integer.

OUTPUT PARAMETERS

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

USAGE

FORTRAN identifiers may be no longer than a maximum of 32 characters. In FORTRAN programs, the name of this routine must be shortened to 32 characters as illustrated:

```
                                 |
12345678901234567890123456789012 | 34567890
---------------------------------|---------
GM_$VIEWPORT_SET_BACKGROUND_VALU | E
```

Use GM_$VIEWPORT_INQ_BACKGROUND_VALUE to retrieve the background value of the specified viewport.

GM_$VIEWPORT_SET_BORDER_SIZE

Specifies the border size of the current viewport, in pixels or fraction-of-bitmap coordinates.

FORMAT

```
GM_$VIEWPORT_SET_BORDER_SIZE (border_unit, border_size, status)
```

INPUT PARAMETERS

border_unit
The units for border size, in GM_$BORDER_UNIT_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$FRACTIONS	Expresses edge width as fractions of the total GM bitmap size.
GM_$PIXELS	Default border type. Expresses edge width in pixels.

border_size
The size of the border, specified as left, bottom, right, top. This is an array of four real values (left, bottom, right, top).

The default border type is in pixels, and the default width is 1,1,1,1.

OUTPUT PARAMETERS

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

USAGE

Viewport borders are drawn with color value 1 for compatibility with monochrome nodes. For the same reason, the graphics metafile package sets the color map for color value 1 to white. With a color node, you may want to use the viewport background color to differentiate viewports from the overall display or the window background. Changing the color map to black is usually not practical because the cursor is also set to color value 1. An alternative is to create the viewport, set the border width to 0 pixels, and then refresh the viewport.

GM_$VIEWPORT_SET_BORDER_SIZE sets the size of the four edges of the current viewport. If border_unit = GM_$PIXELS, edge widths are expressed in pixels. If border_unit = GM_$FRACTIONS, edge widths are expressed as fractions of the total GM bitmap size.

Use GM_$VIEWPORT_INQ_BORDER_SIZE to retrieve the size of the border.

GM_$VIEWPORT_SET_BOUNDS

Changes the display bounds for the current viewport.

FORMAT

```
GM_$VIEWPORT_SET_BOUNDS (bounds, status)
```

INPUT PARAMETERS

bounds

The bounds of the new viewport, in GM_$BOUNDSREAL_T format. This is a four-element array of real values (xmin, ymin, xmax, ymax). See the GM_$ Data Types section for more information.

OUTPUT PARAMETERS

status

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

USAGE

GM_$VIEWPORT_SET_BOUNDS sets the bounds of the current viewport. You must provide two diagonally opposite corners. Coordinates are expressed as fractions of the total display bitmap size: bottom left = (0.0, 0.0); top right = (1.0, 1.0).

Currently, viewports may not overlap.

Use this procedure to change the bounds of the current viewport to fill only the left half of the screen.

```
bounds.xmin := 0.0; bounds.ymin := 0.0;
bounds.xmax := 0.5; bounds.ymax := 1.0;
GM_$VIEWPORT_SET_BOUNDS (bounds, status);
```

GM_$VIEWPORT_SET_GRIDS

Specifies the number and type of grids for a viewport.

FORMAT

```
GM_$VIEWPORT_SET_GRIDS (flags, sindex, cnt, grid, status)
```

INPUT PARAMETERS

flags
Attributes of the snap grid, in GM_$GRID_FLAGS_T format. This is a 2-byte integer. Snapping is not currently implemented. The 2D GMR software ignores this parameter.

sindex
Index of the snap grid. This is a 2-byte integer. Snapping is not currently implemented. The 2D GMR software ignores this parameter.

cnt
Number of grids. This is a 2-byte integer between 1 and GM_$MAX_GRID.

grid
Array of grid descriptions, in GM_$GRID_ARRAY_T format. This is an array of [1 .. cnt] of GM_$GRID_T. See the GM Data Types section for more information.

OUTPUT PARAMETERS

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

USAGE

Snapping has not been implemented for grids.

Grids are input aids; they are not part of the metafile -- they are strictly attributes of a viewport.

A given viewport may have an array of grids associated with it (up to 4 grids, currently). This allows the user to define major and minor grids, or even to define more complicated grids.

Use GM_$VIEWPORT_INQ_GRID to inquire grids for the current viewport.

To determine grid density use GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL.

A call to GM_$VIEWPORT_SET_GRIDS does not actually draw the grids. To display the grids call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH.

GM_$VIEWPORT_SET_REFRESH_STATE

Sets the refresh state of the current viewport.

FORMAT

```
GM_$VIEWPORT_SET_REFRESH_STATE (refresh_state, status )
```

INPUT PARAMETERS

refresh_state

The refresh state of the viewport, in GM_$VIEW_REFRESH_T format. This is a 2-byte integer. Specify only one of the following predefined values:

GM_$REFRESH_INHIBIT

In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH, or when the display is refreshed as the result of a DM command which causes the window to be redrawn. Thus, calling GM_$DISPLAY_REFRESH does not affect a viewport in this refresh state.

GM_$REFRESH_WAIT

(Default) In borrow mode, changing commands in the file does not immediately affect this viewport. The viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH. In direct mode, the viewport is rewritten only when you call GM_$VIEWPORT_REFRESH or GM_$DISPLAY_REFRESH or when the display is refreshed as the result of a DM command which causes the window to be redrawn.

GM_$REFRESH_PARTIAL

Every time you change any command in the file, the following occurs if this viewport is the current viewport: Inserted primitive commands are added, and deleted primitive commands are erased, but underlying data is not rewritten. This provides faster interactive drawing. You should, however, periodically clean up the accumulating inaccuracies by calling GM_$VIEWPORT_REFRESH to redisplay the viewport.

Partial refresh does not always update the viewport accurately. For accuracy in incremental updating, use GM_$REFRESH_UPDATE. Extensive use of partial refresh may necessitate use of GM_$VIEWPORT_REFRESH.

GM_$REFRESH_UPDATE

Every time you change any command in the file, this viewport is completely corrected.

OUTPUT PARAMETERS

status

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

GM_$VISIBLE_INQ_MASK

Returns the value of the visible mask.

FORMAT

```
GM_$VISIBLE_INQ_MASK (mask, status)
```

OUTPUT PARAMETERS

mask
The visible mask value. This is a 4-byte integer.

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

USAGE

Use GM_$VISIBLE_SET_MASK to change the current value of the visible mask.

GM_$VISIBLE_INQ_THRESHOLD

Returns the value of the visible threshold.

FORMAT

```
GM_$VISIBLE_INQ_THRESHOLD (threshold, status)
```

OUTPUT PARAMETERS

threshold
The visible threshold value. This is a 4-byte integer.

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

USAGE

Use GM_$VISIBLE_SET_THRESHOLD to change the current value of the visible threshold.

GM_$VISIBLE_SET_MASK

Sets the value of the visible mask.

FORMAT

```
GM_$VISIBLE_SET_MASK (mask, status)
```

INPUT PARAMETERS

mask

The visible mask value. This is a 4-byte integer.

The visible mask is initialized to 16#7FFFFFFF (all nonzero segments visible).

The visible mask is BIT-ANDed with the segment visible number. If the result is nonzero, the segment may be visible. Both the visible mask and visible threshold must be satisfied for a segment to be visible.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$VISIBLE_INQ_MASK to retrieve the current value of the visible mask.

GM_$VISIBLE_SET_THRESHOLD

Sets the visible threshold.

FORMAT

```
GM_$VISIBLE_SET_THRESHOLD (threshold, status)
```

INPUT PARAMETERS

threshold

The visible threshold value. This is a 4-byte integer.

The visible threshold is initialized to 1 (all nonzero segments visible).

If the segment visible number is greater than or equal to the visible threshold, the segment may be visible. Both the visible mask and visible threshold must be satisfied for a segment to be visible.

OUTPUT PARAMETERS

status

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

USAGE

Use GM_$VISIBLE_INQ_THRESHOLD to retrieve the current value of the visible threshold.

Chapter 3
Errors

This chapter lists 2D GMR errors.

ERRORS

GM_$ABLOCK_ID_INVALID
The ablock identification number is not valid. Use the number assigned when you created the ablock.

GM_$ABLOCK_NOT_CREATED
You must create an ablock before you can use it.

GM_$ACLASS_ID_INVALID
The aclass identification number you used is not valid.

GM_$ALREADY_INITIALIZED
You initialize the GM package only once during a session of using it.

GM_$ANOTHER_SEGMENT_IS_OPEN
Only one segment may be open at a time.

GM_$ATTRIBUTE_VALUE_INVALID
The attribute value is not valid.

GM_$BOUNDS_INVALID
The bounds specified for displaying part of a segment or file do not satisfy the requirement that the minimum value be less than the maximum value.

GM_$CANT_DELETE_FONT_FAMILY_IN_USE
You cannot delete a font family that is referenced by the current file.

GM_$CANT_DELETE_INSTANCED_SEGMENT
You cannot delete a segment if it is instanced by other segments in the file.

GM_$COMMAND_TYPE_DOESNT_MATCH
The current command does not match the type specified in inquire operation.

GM_$COORDINATE_CONVERSION_OVERFLOW
You have supplied a value to a coordinate conversion routine, GM_$COORD..., which cannot be converted.

GM_$DATA_COERCE_NEEDED
To use the data in the format you have supplied, you must convert it.

GM_$FILE_ID_INVALID
The file identification number you used is not valid.

GM_$FILE_NAME_NOT_FOUND
The file name you gave is not valid.

GM_$FONT_FAMILY_ID_INVALID
When you reference a font family, you must use the font family id.

GM_$FONT_FAMILY_NAME_ALREADY_USED
: You may not rename a font family to have the same name as another font family.

GM_$FONT_FAMILY_NAME_NOT_FOUND
: You must include a font family before you can use it.

GM_$ILLEGAL_VALUE
: One of the input parameters that you supplied to the GM package has an illegal value.

GM_$ILLEGAL_SELF_INSTANCE
: A segment may not instance itself, directly or indirectly.

GM_$INPUT_EVENT_TYPE_INVALID
: You must use the event types associated with input routines.

GM_$INVALID_POLYLINE_OPTIONS
: The only options for a polyline are open, closed, or closed and filled. When you specify filled, you must also specify closed.

GM_$NAME_LENGTH_INVALID
: The limitation on the number of characters in a name is 12.

GM_$MODULE_CODE
: GM module

GM_$NEGATIVE_CIRCLE_RADIUS
: The radius of a circle must be a positive value.

GM_$NO_CURRENT_COMMAND
: For editing procedures such as picking, you must have a current command.

GM_$NO_CURRENT_FILE
: You must have a file open. If you have more than one file open and you close the current file, you must select another current file.

GM_$NO_CURRENT_SEGMENT
: You must create or open a segment.

GM_$NO_CURRENT_VIEWPORT
: You must have a current viewport. The current viewport is the last viewport created or selected.

GM_$NO_FONT_FAMILY_INCLUDED
: You must include a font family before you can use it. See GM_$FONT_FAMILY_INCLUDE.

GM_$NO_GM_BITMAP_EXISTS
: When you initialize the GM package, the bitmap size is not defined. The procedure GM_$INQ_BITMAP_SIZE cannot return a valid value until you define the size.

GM_$NO_PICK_MATCHES_FOUND
: The command or segment that you searched for was not found.

GM_$NOTHING_DISPLAYED_IN_VIEWPORT
: You must have displayed a segment in the specified or current viewport before calling this routine.

GM_$NOT_INITIALIZED
: You must initialize the GM package before you can use it.

GM_$OPERATION_OK
Normal status

GM_$PICK_LIST_EMPTY
Only picked segments are included on the pick list. Use GM_$PICK_SEGMENT to list a segment.

GM_$PICK_LIST_NOT_INITIALIZED
To use a pick list, you must first initialize it.

GM_$PICK_LIST_TOO_LONG
The limitation on the number of segments is 32 in a pick list.

GM_$SEGMENT_ID_INVALID
The segment identification number you used is not valid.

GM_$SEGMENT_LOCKED_BY_PICK
You may not delete or edit a segment included in a list of picked segments.

GM_$SEGMENT_NAME_ALREADY_USED
Each segment name must be unique.

GM_$SEGMENT_NAME_NOT_FOUND
The segment name is not in the file.

GM_$TOO_MANY_ABLOCKS
The limitation on the number of ablocks is 40.

GM_$TOO_MANY_FILES
The number of files is limited to 16.

GM_$TOO_MANY_FONT_FAMILIES
The limitation on the number of font families is 8.

GM_$TOO_MANY_SEGMENTS
The limitation on the number of segments is 16384.

GM_$TOO_MANY_VIEWPORTS
The limitation on the number of viewports is 64.

GM_$VIEWPORT_BOUNDS_INVALID
Viewports may not overlap. Space outside of viewports is empty.

GM_$VIEWPORT_DOESNT_EXIST
You must create a viewport before you can use it.

GM_$VIEWPORT_ID_INVALID
You must use the viewport number assigned by GM_$VIEWPORT_CREATE.

GM_$WRONG_DISPLAY_MODE
Each display mode has its advantages and limitations. See GM_$INIT.

Chapter 4
Quick Reference

This section provides a quick reference to 2D GMR routines. Information is presented in two parts: a list organized by function followed by an alphabetical list of calls with their formats.

2D GMR Routines

The following is a list of routines organized by functional category. Some routines are included in more than one category. The method of organization is similar to that in *Programming With DOMAIN 2D Graphics Metafile Resource.*

Developing Application Programs

Controlling the 2D GMR Package

GM_$INIT
: Initializes the graphics metafile package and opens the display.

GM_$TERMINATE
: Terminates the graphics metafile package and closes the display.

Controlling Files

GM_$FILE_CREATE
: Creates a new graphics metafile and makes it the current file.

GM_$FILE_OPEN
: Reopens an existing file and makes it the current file.

GM_$FILE_CLOSE
: Closes the current file, saving revisions or not.

GM_$FILE_SELECT
: Makes the specified file the current file.

Controllling Segments

GM_$SEGMENT_CREATE
: Creates a new segment.

GM_$SEGMENT_OPEN
: Reopens an existing segment.

GM_$SEGMENT_INQ_ID
: Returns the segment identification and the number of instances of the named segment.

GM_$SEGMENT_INQ_CURRENT
: Returns the name, segment identification, and number of instances of the current segment.

GM_$SEGMENT_INQ_NAME
Returns the name of the segment with the specified segment identification number.

GM_$SEGMENT_INQ_COUNT
Returns the number of segments in this metafile and a segment number guaranteed to be greater than or equal to the largest segment number.

GM_$SEGMENT_RENAME
Renames an existing segment.

GM_$SEGMENT_CLOSE
Closes the current segment, saving revisions or not.

GM_$SEGMENT_DELETE
Deletes the current segment.

Primary Segment

GM_$FILE_SET_PRIMARY_SEGMENT
Changes the segment number assumed to be the start of the current file.

Using Basic Modeling Routines

Using Draw and Fill Primitives

GM_$POLYLINE_2D[16,32,REAL]
Inserts a command into the current segment: draw a linked set of line segments.

GM_$RECTANGLE_[16,32,REAL]
Inserts a command into the current segment : draw a rectangle with sides parallel to the x and y axes.

GM_$CIRCLE_[16,32,REAL]
Inserts a command into the current segment: draw a circle.

GM_$CURVE_2D[16,32,REAL]
Inserts a command into the current segment: draw a curve.

GM_$PRIMITIVE_2D[16,32,REAL]
Inserts a command into the current segment: draw a primitive.

Displaying Files and Segments

GM_$DISPLAY_FILE
Displays the entire current file in the current viewport.

GM_$DISPLAY_SEGMENT
Displays the specified segment (and all called segments) in the current viewport.

Displaying Part of a File/Segment

GM_$DISPLAY_FILE_PART
  Displays part of the current file in the current viewport.

GM_$DISPLAY_SEGMENT_PART
  Displays part of the specified segment (and all called segments) in the current viewport.

Using Transformations

GM_$INSTANCE_TRANSLATE_2D[16,32,REAL]
  Inserts a command into the current segment: instance the identified segment with the specified translation.

GM_$INSTANCE_SCALE_2D[16,32,REAL]
  Inserts a command into the current segment: instance the specified segment with the specified scale and translation parameters.

Instances with Arbitrary Transformations

GM_$INSTANCE_TRANSFORM_2D[16,32,REAL]
  Inserts a command to instance the specified segment with the specified rotation and translation applied.

Using Draw and Fill Attributes

Line Attributes

GM_$DRAW_VALUE
  Inserts a command into the current segment: set the value used when drawing lines.

GM_$DRAW_STYLE
  Inserts a command into the current segment: set the line style (solid, dotted).

Fill Attributes

GM_$FILL_VALUE
  Inserts a command into the current segment: set the value used when filling an area.

GM_$FILL_BACKGROUND_VALUE
  Inserts a command into the current segment: set the value used for pixels not in the fill pattern when filling an area.

GM_$FILL_PATTERN
  Inserts a command into the current segment: set the pattern used for the interior of filled areas.

Using Color Map Attributes Using Plane Masks

GM_$PLANE_MASK
  Inserts a command into the current segment: change the plane mask.

Raster Operation Attributes

GM_$DRAW_RASTER_OP
: Inserts a command into the current segment: change the logical raster operations to be performed when drawing.

Using Modeling Routines: Text

Inserting Text

GM_$TEXT_2D[16,32,REAL]
: Inserts a command into the current segment: write a text string.

Using Text Attributes

GM_$TEXT_VALUE
: Inserts a command into the current segment: set the value used when writing text.

GM_$TEXT_BACKGROUND_VALUE
: Inserts a command into the current segment: change the background value used when writing text.

GM_$TEXT_SIZE
: Inserts a command into the current segment : use a different text size from the same font family.

GM_$FONT_FAMILY
: Inserts a command into the current segment: set the font family used when writing text.

Font Families

GM_$FONT_FAMILY_INCLUDE
: Specifies a font family to use in this metafile.

GM_$FONT_FAMILY_INQ_ID
: Returns the identification number of a previously included font family.

GM_$FONT_FAMILY_INQ_NAME
: Returns the font family name for the specified identification number of a previously included font family.

GM_$FONT_FAMILY_RENAME
: Changes the font family file corresponding to this font family identification.

GM_$FONT_FAMILY_EXCLUDE
: Undoes the inclusion of a font family.

Using Segment Characteristics

Primary Segment

GM_$FILE_SET_PRIMARY_SEGMENT
Changes the segment number assumed to be the start of the current file.

GM_$FILE_INQ_PRIMARY_SEGMENT
Returns the segment number assumed to be the start of the current file.

Setting Segment Characteristics

GM_$SEGMENT_SET_VISIBLE
Assigns a visible value to the specified segment.

GM_$SEGMENT_INQ_VISIBLE
Returns the visible value of the specified segment.

GM_$SEGMENT_SET_PICKABLE
Assigns a pickable value to the specified segment.

GM_$SEGMENT_INQ_PICKABLE
Returns the pickable value of the specified segment.

GM_$SEGMENT_SET_TEMPORARY
Makes the specified segment temporary or not. Temporary segments are deleted when the file is closed.

GM_$SEGMENT_INQ_TEMPORARY
Returns whether the specified segment is temporary or not.

Coordinate Data Types

GM_$DATA_COERCE_SET_REAL
Specifies the data type to which subsequent real coordinates are converted.

GM_$DATA_COERCE_INQ_REAL
Returns the data type to which real coordinates are converted.

The Displaying Process

Hardware and Coordinate Systems

GM_$INQ_CONFIG
Returns the current configuration of the display device.

GM_$INQ_BITMAP_SIZE
Returns the size of the GM bitmap in pixels.

GM_$COORD_SEG_TO_BITMAP_2D
Converts segment coordinates to bitmap coordinates.

GM_$COORD_BITMAP_TO_SEG_2D
Converts bitmap coordinates to segment coordinates.

GM_$COORD_PIXEL_TO_SEG_2D
Converts GPR bitmap coordinates used in within-GPR mode to segment coordinates, using a specified transformation.

GM_$COORD_SEG_TO_PIXEL_2D
Converts within-GPR segment coordinates to GPR bitmap coordinates, using a specified transformation.

GM_$COORD_BITMAP_TO_PIXEL_2D
Converts fraction of GM bitmap coordinates to pixel coordinates.

GM_$COORD_PIXEL_TO_BITMAP_2D
Converts pixel to fraction of GM bitmap coordinates.

GM_$VIEWPORT_SEG_2D_TO_PIXEL
Converts segment coordinates to pixel coordinates.

GM_$VIEWPORT_PIXEL_TO_SEG_2D
Converts pixel coordinates to segment coordinates.

Using Multiple Viewports

GM_$VIEWPORT_CLEAR
Clears the current viewport.

GM_$VIEWPORT_CREATE
Creates an additional viewport and makes it the current viewport.

GM_$VIEWPORT_SET_BOUNDS
Changes the display bounds for the current viewport.

GM_$VIEWPORT_INQ_BOUNDS
Returns the bounds of the current viewport.

GM_$VIEWPORT_SELECT
Makes a viewport the current viewport.

GM_$VIEWPORT_DELETE
Deletes a viewport.

GM_$VIEWPORT_INQ_CURRENT
Returns the number of the current viewport.

GM_$VIEWPORT_MOVE
Translates the current viewport, carrying the view with it.

GM_$VIEWPORT_SET_BORDER_SIZE
Specifies the border size of the current viewport, in pixels or fraction-of-bitmap coordinates.

GM_$VIEWPORT_INQ_BORDER_SIZE
Returns the border size of the current viewport, in pixels or fraction-of-bitmap coordinates.

Segment Visibility Criteria

GM_$VISIBLE_SET_MASK
Sets the value of the visible mask.

GM_$VISIBLE_INQ_MASK
Returns the value of the visible mask.

GM_$VISIBLE_SET_THRESHOLD
Sets the visible threshold.

GM_$VISIBLE_INQ_THRESHOLD
Returns the value of the visible threshold.

Display a File/Segment

GM_$DISPLAY_FILE
Displays the entire current file in the current viewport.

GM_$DISPLAY_SEGMENT
Displays the specified segment (and all called segments) in the current viewport.

GM_$DISPLAY_SEGMENT_GPR_2D
In within-GPR mode, allows you to display a segment within a GPR bitmap.

Displaying Part of a File/Segment

GM_$DISPLAY_FILE_PART
Displays part of the current file in the current viewport.

GM_$DISPLAY_SEGMENT_PART
Displays part of the specified segment (and all called segments) in the current viewport.

Changing the View

GM_$VIEW_TRANSLATE
Translates the view under the current viewport.

GM_$VIEW_SCALE
Scales the view under the current viewport, keeping the specified point fixed.

GM_$VIEW_TRANSFORM
Rotates the view under the current viewport, keeping the specified point (in fraction-of-bitmap coordinates) fixed.

GM_$VIEW_TRANSFORM_RESET
Resets the view tranformation to the form in which it was initially displayed.

Refreshing the Display

GM_$DISPLAY_REFRESH
Redisplays all uninhibited viewports of the display.

GM_$VIEWPORT_REFRESH
Refreshes the current viewport.

GM_$REFRESH_SET_ENTRY
Specifies a user-defined routine to be called when the display is refreshed as a result of a DM refresh window or POP command.

Developing Interactive Applications

Changing the Picture

GM_$MODELCMD_SET_MODE
Sets the modeling command mode.

GM_$MODELCMD_INQ_MODE
Returns the values stored for the current (GM_$MODELCMD_SET_MODE) command.

Routines for Interactive Applications

Editing Modes

GM_$MODELCMD_SET_MODE
Sets the modeling command mode.

GM_$MODELCMD_INQ_MODE
Returns the values stored for the current (GM_$MODELCMD_SET_MODE) command.

GM_$REPLACE_SET_FLAG
Sets or clears a flag which causes subsequent commands to replace the current command rather than being inserted after it (Obsolete). New programs use GM_$MODELCMD_SET_MODE.

GM_$REPLACE_INQ_FLAG
Returns the current value of the replace flag (Obsolete). New programs use GM_$MODELCMD_INQ_MODE.

Establishing a Refresh State

GM_$VIEWPORT_SET_REFRESH_STATE
Sets the refresh state of the current viewport.

GM_$VIEWPORT_INQ_REFRESH_STATE
Returns the refresh state of the current viewport.

Controlling the Cursor

GM_$CURSOR_SET_ACTIVE
Specifies whether or not the cursor is displayed.

GM_$CURSOR_SET_PATTERN
Specifies a cursor pattern, type, and origin.

GM_$CURSOR_SET_POSITION
Moves the cursor on the screen.

GM_$CURSOR_INQ_ACTIVE
Returns the status of the cursor: displayed or not displayed.

GM_$CURSOR_INQ_PATTERN
Returns the type, pattern, and origin of the cursor.

GM_$CURSOR_INQ_POSITION
Returns the position of the cursor.

Using Input Operations

GM_$INPUT_ENABLE
Enables an input event type.

GM_$INPUT_DISABLE
Disables an input event type.

GM_$INPUT_EVENT_WAIT
Checks for or waits until an occurrence of an enabled input event.

Setting the Pick Aperture

GM_$PICK_SET_CENTER
Changes the center of the pick aperture.

GM_$PICK_SET_SIZE
Specifies the size of the pick aperture.

GM_$PICK_INQ_CENTER
Returns the center of the pick aperture.

GM_$PICK_INQ_SIZE
Returns the size of the pick aperture.

Picking and Listing Segments

GM_$PICK_SEGMENT
Selects a segment which contains a specified point on the display.

GM_$PICK_INQ_LIST
Returns the current list of picked segments.

GM_$PICK_HIGHLIGHT_SEGMENT
Within the current file, highlights the specified segment.

GM_$PICK_TRANSFORM_POINT
Transforms the coordinates of a point from the coordinate system of the viewport segment to the coordinate system of the picked segment.

Picking a Command

GM_$PICK_COMMAND
Within the current segment, selects a command which contains a selected point on the display.

GM_$PICK_HIGHLIGHT_COMMAND
Highlights the current command on the display.

Controlling What Is Picked

GM_$PICK_SET_THRESHOLD
Sets the value of the threshold used in pick search operations in the current segment.

GM_$PICK_INQ_THRESHOLD
Returns the value of the threshold used in pick search operations in the current segment.

GM_$PICK_SET_MASK
Changes the value of the mask used for segment pickable values during pick segment operations.

GM_$PICK_INQ_MASK
Returns the value of the mask used for segment pickable values during pick segment operations.

Deleting and Copying

GM_$COMMAND_DELETE
Deletes the current command.

GM_$SEGMENT_ERASE
Deletes all commands in the current segment.

GM_$SEGMENT_COPY
Copies the entire contents of another segment into the current segment.

Reading Commands

GM_$INQ_ACLASS
Returns the value stored for the current (GM_$ACLASS) command.

GM_$INQ_CIRCLE_[16,32,REAL]
Returns the values stored for the current (GM_$CIRCLE) command.

GM_$INQ_COMMAND_TYPE
: Returns the command type and the data type of the current command in the current segment.

GM_$INQ_CURVE_2D[16,32,REAL]
: Returns the values stored for the current (GM_$CURVE) command.

GM_$INQ_DRAW_RASTER_OP
: Returns the values stored for the current (GM_$DRAW_RASTER_OP) command.

GM_$INQ_DRAW_STYLE
: Returns the values stored for the current (GM_$DRAW_STYLE) command.

GM_$INQ_DRAW_VALUE
: Returns the value stored for the current (GM_$DRAW_VALUE) command.

GM_$INQ_FILL_BACKGROUND_VALUE
: Returns the value stored for the current (GM_$FILL_BACKGROUND_VALUE) command.

GM_$INQ_FILL_PATTERN
: Returns the value stored for the current (GM_$FILL_PATTERN) command.

GM_$INQ_FILL_VALUE
: Returns the value stored for the current (GM_$FILL_VALUE) command.

GM_$INQ_FONT_FAMILY
: Inserts a command into the current segment: set the font family used when writing text.

GM_$INQ_INSTANCE_SCALE_2D[16,32,REAL]
: Returns the value stored for the current (GM_$INSTANCE_SCALE_2D) command.

GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL]
: Returns the value stored for the current (GM_$INSTANCE_TRANSFORM) command.

GM_$INQ_INSTANCE_TRANSLATE_2D[16,32,REAL]
: Returns the value stored for the current (GM_$INSTANCE_TRANSLATE_2D) command.

GM_$INQ_PLANE_MASK
: Returns the value stored for the current (GM_$PLANE_MASK) command.

GM_$INQ_POLYLINE_2D[16,32,REAL]
: Returns the values stored for the current (GM_$POLYLINE_2D) command.

GM_$INQ_PRIMITIVE_2D[16,32,REAL]
: Returns the values stored for the current (GM_$PRIMITIVE) command.

GM_$INQ_RECTANGLE_[16,32,REAL]
: Returns the values stored for the current (GM_$RECTANGLE) command.

GM_$INQ_TAG
: Returns the value stored for the current (GM_$TAG) command.

GM_$INQ_TEXT_2D[16,32,REAL]
Returns the value stored for the current (GM_$TEXT_2D[16,32,REAL]) command.

GM_$INQ_TEXT_BACKGROUND_VALUE
Returns the value stored for the current (GM_$TEXT_BACKGROUND_VALUE) command.

GM_$INQ_TEXT_SIZE
Returns the value stored for the current (GM_$TEXT_SIZE) command.

GM_$INQ_TEXT_VALUE
Returns the value stored for the current (GM_$TEXT_VALUE) command.

Using Within-GPR Mode

Displaying User-Defined Primitives

GM_$PRIMITIVE_DISPLAY_2D
Assigns the specified user-defined routine to the specified user-defined primitive type number.

Output

Printing

GM_$PRINT_FILE
Converts the current metafile to the specified file for subsequent printing on a hard-copy device.

GM_$PRINT_FILE_PART
Converts part of the current metafile to the specified file for subsequent printing on a hard-copy device.

Attribute Classes and Blocks

Using Attribute Classes

GM_$ACLASS
Inserts a command into the current segment: change to a different attribute class.

Creating Attribute Blocks

GM_$ABLOCK_CREATE
Creates an attribute block and initializes it equivalent to an existing block.

Modifying Attribute Blocks

GM_$ABLOCK_SET_DRAW_RASTER_OP
Changes the raster operation code for drawing lines for this attribute block.

GM_$ABLOCK_SET_DRAW_STYLE
Changes the value of the line style in this attribute block.

GM_$ABLOCK_SET_DRAW_VALUE
Changes the value for drawing lines in this attribute block.

GM_$ABLOCK_SET_FILL_PATTERN
Changes the fill pattern in this attribute block.

GM_$ABLOCK_SET_FILL_VALUE
Changes the value for filling areas in this attribute block.

GM_$ABLOCK_SET_PLANE_MASK
Changes the value of the plane mask in this attribute block.

GM_$ABLOCK_SET_TEXT_VALUE
Changes the value for writing text set for this attribute block.

GM_$ABLOCK_SET_TEXT_BACKGROUND
Changes the background value for text in this attribute block.

GM_$ABLOCK_SET_TEXT_SIZE
Changes the size of text in this attribute block.

GM_$ABLOCK_SET_FONT_FAMILY
Changes the font family in this attribute block.

Reading Attribute Blocks

GM_$ABLOCK_INQ_DRAW_RASTER_OP
Returns the raster operation code for drawing lines for the specified attribute block.

GM_$ABLOCK_INQ_DRAW_STYLE
Returns the line style set for the specified attribute block.

GM_$ABLOCK_INQ_DRAW_VALUE
Returns the value for drawing lines set for the specified attribute block.

GM_$ABLOCK_INQ_FILL_PATTERN
Returns the pattern set for filling areas for the specified attribute block.

GM_$ABLOCK_INQ_FILL_VALUE
Returns the value set for filling areas for the specified attribute block.

GM_$ABLOCK_INQ_PLANE_MASK
Returns the value of the plane mask set for the specified attribute block.

GM_$ABLOCK_INQ_TEXT_VALUE
Returns the value for writing text for the specified attribute block.

GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE
Returns the text background value set for the specified attribute block.

GM_$ABLOCK_INQ_TEXT_SIZE
Returns the size of text set for the specified attribute block.

GM_$ABLOCK_INQ_FONT_FAMILY
Returns the font family identification number set for the specified attribute block.

Copying Attribute Blocks

GM_$ABLOCK_COPY
Copies all attributes from one existing attribute block to another.

Attributes and Viewing Operations

GM_$ABLOCK_ASSIGN_DISPLAY
Assigns an attribute block (by number) to an attribute class, for the entire display.

GM_$ABLOCK_INQ_ASSIGN_DISPLAY
Returns the current attribute block number assigned to a particular attribute class for the display.

GM_$ABLOCK_ASSIGN_VIEWPORT
Assigns an attribute block (by number) to an attribute class, for one viewport.

GM_$ABLOCK_INQ_ASSIGN_VIEWPORT
Returns the current attribute block number assigned to a particular attribute class for one viewport.

Advanced Display Techniques

Changing the Color Map

GM_$DISPLAY_SET_COLOR_MAP
Changes values in the display color map.

GM_$DISPLAY_INQ_COLOR_MAP
Returns the values in the display color map.

Using Viewport Techniques

GM_$VIEWPORT_SET_BACKGROUND_VALUE
Sets the pixel value used for the background of the specified viewport.

GM_$VIEWPORT_INQ_BACKGROUND_VALUE
Returns the pixel value used for the background of the specified viewport.

Programming Techniques

Using Tags

GM_$TAG
Inserts a comment into the current segment.

GM_$TAG_LOCATE
Looks for the specified tag in the specified range of segments and returns the segment ID of the lowest numbered segment in which the tag is found.

Format for User-Callable Routines: Alphabetical Listing

```
GM_$ABLOCK_ASSIGN_DISPLAY (aclass_id, ablock_id, status)

GM_$ABLOCK_ASSIGN_VIEWPORT (aclass_id, viewport_id, ablock_id, status)

GM_$ABLOCK_COPY (source_ablock_id, destination_ablock_id, status)

GM_$ABLOCK_CREATE (source_ablock_id, ablock_id, status)

GM_$ABLOCK_INQ_ASSIGN_DISPLAY (aclass_id, ablock_id, status)

GM_$ABLOCK_INQ_ASSIGN_VIEWPORT (aclass_id, viewport_id, ablock_id, status)

GM_$ABLOCK_INQ_DRAW_RASTER_OP (ablock_id, raster_op, status)

GM_$ABLOCK_INQ_DRAW_STYLE (ablock_id, style, repeat_factor, pattern,
                           pattern_length, status)

GM_$ABLOCK_INQ_DRAW_VALUE (ablock_id, value, status)

GM_$ABLOCK_INQ_FILL_BACKGROUND_VALUE (ablock_id, value, status)

GM_$ABLOCK_INQ_FILL_PATTERN (ablock_id, scale, size, pattern, status)

GM_$ABLOCK_INQ_FILL_VALUE (ablock_id, value, status)

GM_$ABLOCK_INQ_FONT_FAMILY (ablock_id, font_family_id, status)

GM_$ABLOCK_INQ_PLANE_MASK (ablock_id, change, mask, status)

GM_$ABLOCK_INQ_TEXT_BACKGROUND_VALUE (ablock_id, value, status)

GM_$ABLOCK_INQ_TEXT_SIZE (ablock_id, size, status)

GM_$ABLOCK_INQ_TEXT_VALUE (ablock_id, value, status)

GM_$ABLOCK_SET_DRAW_RASTER_OP (ablock_id, raster_op, status)

GM_$ABLOCK_SET_DRAW_STYLE (ablock_id, style, repeat_factor, pattern,
                           pattern_length, status)

GM_$ABLOCK_SET_DRAW_VALUE (ablock_id, value, status)

GM_$ABLOCK_SET_FILL_BACKGROUND_VALUE (ablock_id, value, status)

GM_$ABLOCK_SET_FILL_PATTERN (ablock_id, scale, size, pattern, status)

GM_$ABLOCK_SET_FILL_VALUE (ablock_id, value, status)

GM_$ABLOCK_SET_FONT_FAMILY (ablock_id, font_family_id, status)

GM_$ABLOCK_SET_PLANE_MASK (ablock_id, change, mask, status)

GM_$ABLOCK_SET_TEXT_BACKGROUND_VALUE (ablock_id, value,status)
```

```
GM_$ABLOCK_SET_TEXT_SIZE (ablock_id, size, status)

GM_$ABLOCK_SET_TEXT_VALUE (ablock_id, value, status)

GM_$ACLASS (aclass_id, status)

GM_$CIRCLE_16 (center, radius, fill, status)

GM_$CIRCLE_32 (center, radius, fill, status)

GM_$CIRCLE_REAL (center, radius, fill, status)

GM_$COMMAND_DELETE (status)

GM_$COMMAND_INQ_BOUNDS (bounds,status)

GM_$COORD_BITMAP_TO_PIXEL_2D (bitmap_position, pixel_position,status)

GM_$COORD_BITMAP_TO_SEG_2D (bitmap_position, segment_position,status)

GM_$COORD_PIXEL_TO_BITMAP_2D (pixel_position, bitmap_position, status)

GM_$COORD_PIXEL_TO_SEG_2D (rotate, translate, pixel_position,
                          segment_position, status)

GM_$COORD_SEG_TO_BITMAP_2D (segment_position, bitmap_position, status)

GM_$COORD_SEG_TO_PIXEL_2D (rotate, translate, segment_position,
                           pixel_position, status)

GM_$CURSOR_INQ_ACTIVE (active, status)

GM_$CURSOR_INQ_PATTERN (style, pattern_size, pattern, origin, status)

GM_$CURSOR_INQ_POSITION (bitmap_position, status)

GM_$CURSOR_SET_ACTIVE (active, status)

GM_$CURSOR_SET_PATTERN (style, pattern_size, pattern, origin, status)

GM_$CURSOR_SET_POSITION (bitmap_position, status)

GM_$CURVE_2D16 (curve_type, n_points, point_array, n_parameters,
                parameter_array, status)

GM_$CURVE_2D32 (curve_type, n_points, point_array, n_parameters,
                parameter_array, status)

GM_$CURVE_2DREAL (curve_type, n_points, point_array, n_parameters,
                  parameter_array, status)

GM_$DATA_COERCE_INQ_REAL (data_type, status)

GM_$DATA_COERCE_SET_REAL (data_type, status)

GM_$DISPLAY_FILE (status)

GM_$DISPLAY_FILE_PART (bounds, status)
```

```
GM_$DISPLAY_INQ_COLOR_MAP (start_index, n_entries, values, status)

GM_$DISPLAY_REFRESH (status)

GM_$DISPLAY_SEGMENT (segment_id, status)

GM_$DISPLAY_SEGMENT_GPR_2D (segment_id, rotate, translate, status)


GM_$DISPLAY_SEGMENT_PART (segment_id, bounds, status)

GM_$DISPLAY_SET_COLOR_MAP (start_index, n_entries, values, status)

GM_$DRAW_RASTER_OP (raster_op, status)

GM_$DRAW_STYLE (style, repeat_factor, pattern, pattern_length, status)

GM_$DRAW_VALUE (value, status)

GM_$FILE_CLOSE (save, status)

GM_$FILE_COMPACT(name, name_length, status)

GM_$FILE_CREATE (name, name_length, access, concurrency, file_id, status)

GM_$FILE_INQ_BOUNDS (bounds,status)

GM_$FILE_INQ_PRIMARY_SEGMENT (segment_id, status)

GM_$FILE_OPEN (name, name_length, access, concurrency, file_id, status)

GM_$FILE_SELECT (file_id, status)

GM_$FILE_SET_PRIMARY_SEGMENT (segment_id, status)

GM_$FILL_BACKGROUND_VALUE (value, status)

GM_$FILL_PATTERN (scale, size, pattern, status)

GM_$FILL_VALUE (value, status)

GM_$FONT_FAMILY (font_family_id, status)

GM_$FONT_FAMILY_EXCLUDE (font_family_id, status)

GM_$FONT_FAMILY_INCLUDE (pathname, pathname_length, font_type,
                         font_family_id, status)

GM_$FONT_FAMILY_INQ_ID (pathname, pathname_length, font_type,
                        font_family_id, status)

GM_$FONT_FAMILY_INQ_NAME (font_family_id, font_type, pathname,
                          pathname_length, maximum_length, status)

GM_$FONT_FAMILY_RENAME (font_family_id, pathname, pathname_length,
                        font_type, status)

GM_$INIT (display_mode, unit, size, n_planes, status)
```

```
GM_$INPUT_DISABLE (event_type, status)

GM_$INPUT_ENABLE (event_type, key_set, status)

GM_$INPUT_EVENT_WAIT (wait, event_type, event_data, bitmap_position,
                      viewport_id, segment_position, status)

GM_$INQ_ACLASS (aclass_id, status)

GM_$INQ_BITMAP_SIZE (size, planes, status)

GM_$INQ_CIRCLE_16 (center, radius, fill, status)

GM_$INQ_CIRCLE_32 (center, radius, fill, status)

GM_$INQ_CIRCLE_REAL (center, radius, fill, status)

GM_$INQ_COMMAND_TYPE  (command_type, data_type, status)

GM_$INQ_CONFIG (configuration, status)

GM_$INQ_CURVE_2D16 (curve_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$INQ_CURVE_2D32 (curve_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$INQ_CURVE_2DREAL (curve_type, n_points, point_array, n_parameters,
                      parameter_array, status)

GM_$INQ_DRAW_RASTER_OP (raster_op, status)

GM_$INQ_DRAW_STYLE (style, repeat_factor, pattern, pattern_length, status)

GM_$INQ_DRAW_VALUE (value, status)

GM_$INQ_FILL_BACKGROUND_VALUE (value, status)

GM_$INQ_FILL_PATTERN (scale, size, pattern, status)

GM_$INQ_FILL_VALUE (value, status)

GM_$INQ_FONT_FAMILY (font_family_id, status)

GM_$INQ_INSTANCE_SCALE_2D16 (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_SCALE_2D32 (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_SCALE_2DREAL (segment_id, scale, translate, status)

GM_$INQ_INSTANCE_TRANSFORM_2D[16,32,REAL] (segment_id, rotate, translate,
                                           status)

GM_$INQ_INSTANCE_TRANSLATE_2D16 (segment_id, translate, status)

GM_$INQ_INSTANCE_TRANSLATE_2D32 (segment_id,  translate, status)

GM_$INQ_INSTANCE_TRANSLATE_2DREAL (segment_id, translate, status)
```

```
GM_$INQ_PLANE_MASK (mask, status)

GM_$INQ_POLYLINE_2D16 (n_points, point_array, close, fill, status)

GM_$INQ_POLYLINE_2D32 (n_points, point_array, close, fill, status)

GM_$INQ_POLYLINE_2DREAL (n_points, point_array, close, fill, status)

GM_$INQ_PRIMITIVE_2D16  (primitive_type, n_points, point_array,
                         n_parameters, parameter_array, status)

GM_$INQ_PRIMITIVE_2D32 (primitive_type, n_points, point_array,
                        n_parameters, parameter_array, status)

GM_$INQ_PRIMITIVE_2DREAL (primitive_type, n_points, point_array,
                          n_parameters, parameter_array, status)

GM_$INQ_RECTANGLE_16 (point1, point2, fill, status)

GM_$INQ_RECTANGLE_32 (point1, point2, fill, status)

GM_$INQ_RECTANGLE_REAL (point1, point2, fill, status)

GM_$INQ_TAG (string, string_length, status)

GM_$INQ_TEXT_2D[16,32,REAL] (point, rotate, string, string_length, status)

GM_$INQ_TEXT_BACKGROUND_VALUE (value, status)

GM_$INQ_TEXT_SIZE (size, status)

GM_$INQ_TEXT_VALUE (value, status)

GM_$INSTANCE_SCALE_2D16 (segment_id, scale, translate, status)

GM_$INSTANCE_SCALE_2D32 (segment_id, scale, translate, status)

GM_$INSTANCE_SCALE_2DREAL (segment_id, scale, translate, status)

GM_$INSTANCE_TRANSFORM_2D16 (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSFORM_2D32 (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSFORM_2DREAL (segment_id, rotate, translate, status)

GM_$INSTANCE_TRANSLATE_2D16 (segment_id, translate, status)

GM_$INSTANCE_TRANSLATE_2D32 (segment_id, translate, status)

GM_$INSTANCE_TRANSLATE_2DREAL (segment_id, translate, status)

GM_$MODELCMD_INQ_MODE (gm_modelcmd_mode, status)

GM_$MODELCMD_SET_MODE (gm_modelcmd_mode, status)

GM_$PICK_COMMAND (search_rule, status)

GM_$PICK_HIGHLIGHT_COMMAND (highlight, time, status)
```

```
GM_$PICK_HIGHLIGHT_SEGMENT (highlight, time, status)

GM_$PICK_INQ_CENTER (center, status)

GM_$PICK_INQ_LIST (max_length, length, list, status);

GM_$PICK_INQ_MASK (mask, status)

GM_$PICK_SET_SIZE (size, status)

GM_$PICK_INQ_THRESHOLD (threshold, status)

GM_$PICK_SEGMENT (search_rule, segment_id, n_instances, bounds, status)

GM_$PICK_SET_CENTER (center, status)

GM_$PICK_SET_MASK (mask, status)

GM_$PICK_SET_SIZE (size, status)

GM_$PICK_SET_THRESHOLD (threshold, status)

GM_$PICK_TRANSFORM_POINT (vsegment_position, psegment_position, status)

GM_$PLANE_MASK (mask, status)

GM_$POLYLINE_2D16 (n_points, point_array, close, fill, status)

GM_$POLYLINE_2D32 (n_points, point_array, close, fill, status)

GM_$POLYLINE_2DREAL (n_points, point_array, close, fill, status)

GM_$PRIMITIVE_2D16 (primitive_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$PRIMITIVE_2D32 (primitive_type, n_points, point_array, n_parameters,
                    parameter_array, status)

GM_$PRIMITIVE_2DREAL (primitive_type, n_points, point_array, n_parameters,
                      parameter_array, status)

GM_$PRIMITIVE_DISPLAY_2D (primitive_type, display_procedure_ptr, status)

GM_$PRINT_FILE (file_name, file_name_length, size, invert, print_style,
                bpi, status)

GM_$PRINT_FILE_PART (bounds, file_name, file_name_length, size, invert,
                     print_style, bpi, status)

GM_$RECTANGLE_16 (point1, point2, fill, status)

GM_$RECTANGLE_32 (point1, point2, fill, status)

GM_$RECTANGLE_REAL (point1, point2, fill, status)

GM_$REFRESH_SET_ENTRY (refresh_procedure_ptr, status)

GM_$REPLACE_INQ_FLAG (yes_no, status)
```

```
GM_$REPLACE_SET_FLAG (yes_no, status)

GM_$SEGMENT_CLOSE (save, status)

GM_$SEGMENT_COPY (segment_id, status)

GM_$SEGMENT_CREATE (name, name_length, segment_id, status)

GM_$SEGMENT_DELETE (status)

GM_$SEGMENT_ERASE (status)

GM_$SEGMENT_INQ_BOUNDS (bounds,status)

GM_$SEGMENT_INQ_COUNT (count, max_segid, status)

GM_$SEGMENT_INQ_CURRENT (name, name_length, segment_id,
                         n_instances, status)

GM_$SEGMENT_INQ_ID (name, name_length, segment_id, n_instances, status)

GM_$SEGMENT_INQ_NAME (seg_id, name, name_length, n_instances, status)

GM_$SEGMENT_INQ_PICKABLE (segment_id, pickable, status)

GM_$SEGMENT_INQ_TEMPORARY (segment_id, temporary, status)

GM_$SEGMENT_INQ_VISIBLE (segment_id, visible, status)

GM_$SEGMENT_OPEN (segment_id, status)

GM_$SEGMENT_RENAME (segment_id, name, name_length, status)

GM_$SEGMENT_SET_PICKABLE (segment_id, pickable, status)

GM_$SEGMENT_SET_TEMPORARY (segment_id, temporary, status)

GM_$SEGMENT_SET_VISIBLE (segment_id, visible, status)

GM_$TAG (string, string_length, status)

GM_$TAG_LOCATE (string, string_length, min, max, segment_id, status)

GM_$TERMINATE (status)

GM_$TEXT_2D[16,32,REAL] (point, rotate, string, string_length, status)

GM_$TEXT_BACKGROUND_VALUE (value, status)

GM_$TEXT_SIZE (size, status)

GM_$TEXT_VALUE (value, status)

GM_$VIEW_SCALE (scale, point, status)

GM_$VIEW_TRANSFORM (rotate, point, status)

GM_$VIEW_TRANSFORM_RESET (status)
```

```
GM_$VIEW_TRANSLATE (translate, status)

GM_$VIEWPORT_CLEAR (value, status)

GM_$VIEWPORT_CREATE (bounds, viewport_id, status)

GM_$VIEWPORT_DELETE (viewport_id, status)

GM_$VIEWPORT_INQ_BACKGROUND_VALUE (viewport_id, value, status)

GM_$VIEWPORT_INQ_BORDER_SIZE (border_unit, border_size, status)

GM_$VIEWPORT_INQ_BOUNDS (bounds, status)

GM_$VIEWPORT_INQ_CURRENT (viewport_id, status)

GM_$VIEWPORT_INQ_GRIDS (flags, index, cnt, grid, status)

GM_$VIEWPORT_INQ_REFRESH_STATE (refresh_state, status)

GM_$VIEWPORT_MOVE (translate, status)

GM_$VIEWPORT_PIXEL_TO_SEG_2D (pixel_position, viewport_id, segment_position
                              status)

GM_$VIEWPORT_REFRESH (status)

GM_$VIEWPORT_SEG_2D[16,32,REAL]_TO_PIXEL (p, q, status)

GM_$VIEWPORT_SELECT (viewport_id, status)

GM_$VIEWPORT_SET_BACKGROUND_VALUE (viewport_id, value, status)

GM_$VIEWPORT_SET_BORDER_SIZE (border_unit, border_size, status)

GM_$VIEWPORT_SET_BOUNDS (bounds, status)

GM_$VIEWPORT_SET_GRIDS (flags, index, cnt, grid, status)

GM_$VIEWPORT_SET_REFRESH_STATE (refresh_state, status )

GM_$VISIBLE_INQ_MASK (mask, status)

GM_$VISIBLE_INQ_THRESHOLD (threshold, status)

GM_$VISIBLE_SET_MASK (mask, status)

GM_$VISIBLE_SET_THRESHOLD (threshold, status)
```

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 2D Graphics Metafile Resource Call Reference*
Order No.: 009793 Revision: 00 Date of Publication: November, 1986

What type of user are you?

_____ System programmer; language ____________	
_____ Applications programmer; language ____________	
_____ System maintenance person	_____ Manager/Professional
_____ System Administrator	_____ Technical Professional
_____ Student Programmer	_____ Novice
_____ Other	

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