GEM

Programmer's Guide

Volume 1: VDI

174

COPYRIGHT

DISCLAIMER

NOTICE TO USER

From time to time changes are made in the filenames and in the files actually included on the distribution disk. This manual should not be construed as a representation or warranty that such files or facilities exist on the distribution disk or as part of the materials and programs distributed. Most distribution disks include a "README.DOC" file. This file explains variations from the manual which do constitute modification of the manual and the items included therewith. Be sure to read this file before using the software.

TRADEMARKS

The GEM Programmer's Guide, Volume 1: VDI was printed in the United States of America.

****
* First Edition: March 1985 *
****

FOREWORD

OBJECTIVE

This guide describes the features and operation of the Graphics Environment Manager..(GEM..) Virtual Device Interface (VDI), the successor to the Digital Research..Graphics System Extension (GSX..). You can write graphics applications using GEM VDI capabilities.

AUDIENCE

This guide is intended for microcomputer application programmers with operating system and graphics programming experience.

ORGANIZATION

This guide contains nine sections, nine appendixes, a glossary, and an index. The detachable reference card at the end of this guide lists the GEM VDI functions by opcode number and gives their respective C binding procedure names. It also lists the section of this guide in which each function is discussed.

Section 1 introduces GEM VDI. It describes the GEM VDI architecture, including the Graphics Device Operating System (GDOS) and the device drivers.

Section 2 describes GEM VDI operating procedures and how to integrate application programs with GEM VDI.

Section 3 describes the control functions, which initialize the graphics workstation and set defaults for use with the application.

Section 4 describes the output functions, which cause graphics primitives to be displayed on a graphics output device (a screen or plotter, for example).

Section 5 describes the attribute functions, which determine qualities of all subsequent output primitives, such as color and style.

Section 6 describes the raster functions, which perform logic operations on raster areas (rectangular blocks of bits in memory or pixels on physical devices).

Section 7 describes the input functions, which allow the user to interact with the application program.

Section 8 describes the inquire functions, which return the current settings for device-specific attributes, such as the number of text styles supported.

Section 9 describes the escape functions, which allow the application program to access special device capabilities.

Appendix A lists and describes the GEM VDI error messages.

Appendix B explains the ASSIGN.SYS file contents, which include information the GDOS uses to identify the output device.

Appendix C lists and describes the GEM VDI metafile format.

Appendix D defines the GEM VDI standard keyboard.

Appendix E describes the mapping of GEM VDI to specific operating systems and the calling procedures needed to perform that mapping.

Appendix F includes the system fonts.

Appendix G describes the font file format.

Appendix H describes the reserved metafile sub-opcodes.

Appendix I describes the bit image file format.

TABLE OF CONTENTS

178

TABLE OF CONTENTS (continued)

181

182

184

Appendixes

185

TABLE OF CONTENTS (continued)

186

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Figures

Listings

188

Section 1
OVERVIEW

INTRODUCTION

The GEM VDI provides a device-independent environment in which you can write graphics applications. This section describes GEM VDI and its architecture. Subsequent sections describe writing an application and all the GEM VDI functions.

FEATURES

The following features of GEM VDI make it possible for you to write graphics applications that run under several microcomputer operating systems:

- GEM VDI provides a common graphics programming interface that is compatible with the most widely used operating systems, thus making it easy to port many programs.
- GEM VDI provides a device-independent software interface for your application programs. You do not need to rewrite applications for use with different output devices such as screens, printers, and plotters. GEM VDI handles device differences and makes it possible for you to send information to the devices through the application program as if the devices were the same. GEM VDI handles graphics requests and supplies the right driver to run the specific device.

ENHANCEMENTS

GEM VDI includes enhancements to GSX functions and now includes the following capabilities:

- raster functions--functions that affect raster areas, which are rectangular blocks of pixels on physical devices or rectangular blocks of bits in memory
- faces--letter styles stored in dynamically loadable files

ARCHITECTURE

GEM VDI provides graphics primitives for implementing graphics applications with reduced programming effort. Application programs interface to GEM VDI through a standard calling sequence. Drivers for specific graphics devices translate the standard GEM VDI calls to the unique characteristics of each device. In this way, GEM VDI provides device independence.

GEM VDI is composed of two components:

- Graphics Device Operating System (GDOS)
- device drivers and face files

The GDOS contains the device-independent graphics functions, while the device drivers and face files contain the device-dependent code.

GEM VDI is designed in this way to make the principal parts of the GDOS transportable to different hardware configurations. This design also allows applications to run independently of the specific devices connected to the system.

Graphics Device Operating System (GDOS)

The Graphics Device Operating System (GDOS) contains the basic host and device-independent graphics functions that can be called by your application program. GDOS provides a standard graphics interface that is constant regardless of specific devices or host hardware, just as the disk operating system standardizes disk interfaces. Your application program accesses the GDOS in much the same way that it accesses the operating system.

The GDOS performs coordinate scaling so that your application can specify points in a normalized space. It uses device-specific information to transform (map) the coordinates into the corresponding values for a particular graphics device.

An application can also specify points in raster coordinate space, in which case no transformation occurs.

Graphics Device Drivers

The graphics device drivers are similar to any I/O system. They contain the device-specific code required to interface your particular graphics devices to the GDOS. The device drivers communicate directly with the graphics devices. GEM VDI requires a unique device driver for each graphics device in a system.

A single program can use several graphics devices; the GDOS loads only the appropriate device driver file into memory. By referring to devices with a device identification number, an application program can send graphics information to any one of several memory-resident device drivers.

The device driver outputs the GEM VDI graphics primitives according to the inherent capabilities of a particular graphics device. In some cases, a device driver emulates standard capabilities not provided by the graphics device hardware. For example, some devices require that dashed lines be simulated by a series of short vectors generated in the device driver.

The GEM VDI package contains drivers for many of the most popular microcomputer-related graphics devices.

DEVICE TYPES

You can write a GEM VDI-based graphics application for a variety of devices including screens, plotters, printers, and special cameras.

Metafiles

A metafile is the stored generic form of a picture file. Any application can create a GEM VDI metafile that can later be called into another graphics application. The metafile driver stores a description of a picture in a data file. These files can later be sent to any device or used to exchange a picture between two applications.

When GEM VDI creates a metafile, it provides the ideal device. Raster Coordinate (RC) and Normalized Device Coordinate (NDC) space are the same (0 to 32767). No transform is applied. Refer to "Transforming Points" later in this section for more information on the coordinate spaces.

Refer to Appendix C for information about the file format for metafiles.

Multiple Workstations

The application program specifies the graphics function to be performed by a device driver with an operation code (opcode) in the control array. "Opcodes" in Section 2 describes the opcodes.

Because multiple workstations can be open at the same time, each GEM VDI function must be provided with a unique reference to the desired device. This identification is referred to as the device handle.

Device Handles

The GDOS assigns the device handle when the Open Workstation function is called by the application program. The Open Workstation call returns the device handle in the array element contrl(6). All subsequent GEM VDI calls need to supply the device handle as an input in element contrl(6).

ASSIGN.SYS

The ASSIGN.SYS file is a text file, and can be created or edited using any text editor. The file lists the device driver filenames and face filenames, their device numbers, and device-specific information. The device numbers are assigned according to their type. Refer to Table 1-1 for device numbers.

Table 1-1. Device Identification Numbers

Device Type	Device Number
Screen	1-10
Plotter	11-20
Printer	21-30
Metafile	31-40
Camera	41-50
Tablet	51-60

APPLICATION PROGRAMS

With appropriate calls to the GDOS, you can write application programs in assembly language or in a high-level language that supports the GEM VDI calling conventions. You can compile or assemble and link programs containing GEM VDI calls in the normal manner. Refer to Section 2 for more information about writing graphics application programs.

VIRTUAL DEVICE INTERFACE

This guide contains the specification of the GEM Virtual Device Interface (VDI) and defines how applications interface to GEM VDI. The GEM VDI specifies the calling sequence to access device driver functions as well as the necessary calling parameters. Refer to Appendix E for the main entry into the VDI for your operating system.

The main entry point into the VDI is a single subroutine with five arguments, in the form of five arrays:

- o control array
- o array of input parameters
- o array of input point coordinates
- o array of output parameters
- o array of output point coordinates

193

All array elements are of type INTEGER (2 bytes). All arrays are zero-based; that is, the double-word address of the Parameter Block (PB) points to the first element of the control array, contrl(0). The content of the input and output parameter arrays depends on the opcode. Refer to Section 2 for more information about writing graphics applications.

TRANSFORMING POINTS

All computer graphics are displayed using a coordinate system. GEM VDI makes sure the coordinate system of one device matches the coordinate system of another. For example, with GEM VDI, the application program produces the same graphics image on a printer as on a screen. The linetypes and fill styles are the same in Normalized Device Coordinates (NDC), which are described below. Character sizes are different. The same number of characters are displayed per line, but a printer's line length is generally greater than a screen's.

Transformation Mode

The application program can address the display surface using one of two coordinate systems:

- Normalized Device Coordinates (NDC)
- Raster Coordinates (RC)

The transformation mode, specified at Open Workstation, determines which coordinate system is used.

Normalized Device Coordinates

Normalized Device Coordinates (NDC) address the graphics display independent of the device coordinate size. These units are then mapped to Raster Coordinates by the GDOS. The transformation mode set at Open Workstation determines whether the GDOS maps from NDC units to the Raster Coordinates.

The full scale of NDC space (0-32767) is mapped to the full dimensions of the device on both axes. On a nonsquare display with square pixels, a different scale factor is applied to each axis with this transformation mode.

NDC space has its origin at the lower left corner, and its (xmax,ymax) point at the upper right corner. This space is in the first quadrant of the Cartesian coordinate system.

When transforming from NDC to Raster Coordinates (RC), the GDOS assumes a raster coordinate at the bottom left edge of a pixel. You should compensate for a boundary condition created at the top edge of NDC space.

This problem is best illustrated with an example. Given the display of Figure 1-1 in Transformation Mode 0, the NDC point (32767,32767) maps to the point (0,200) in RCs. But because pixels are addressed at their lower left corner, the NDC point (32766,32766) maps to the point (1,199) in RCs. The application programmer should correct for this boundary error by adding half of the NDC height and width into the coordinate transform to ensure that any roundoff error in the application-world-to-NDC transform does not cause the wrong pixel to be addressed.

Raster Coordinates

Raster Coordinates (RC) are actual device units (for example, rasters for screens or steps for plotters and printers). Unlike NDCs, RCs have their origin at the upper left corner, and the (xmax,ymax) point at the bottom right pixel of the space. Refer to Figure 1-1 for an illustration of this concept.

No transformation occurs when the RC system is in effect. The application needs to adjust its transform based on the aspect ratio of pixels on the screen. The raster coordinate system saves the overhead of the GDOS having to perform a transformation on every point.

Figure 1-1. Transformation Modes

End of Section 1

Section 2
WRITING A GRAPHICS APPLICATION

INTRODUCTION

This section explains how to use GEM VDI in your graphics applications.

GEM VDI DISTRIBUTION FILES

When you receive your GEM VDI distribution disks, first duplicate them and then store them in a safe place. Then, using the duplicate disks, transfer the GEM VDI files to working system disks. Always use the duplicate disks to generate any new copies of GEM VDI. Do not use the distribution disks for routine operations.

WRITING THE PROGRAM

You can write your graphics application in one of two ways:

o using assembly language

o using high-level language bindings (C language bindings are provided.)

The first method addresses functions by their opcode numbers, the second by procedure name. The C Language bindings provided for each function allow for portability across implementations. In the C bindings, which appear with each function in sections 3 through 9, WORD declares a 16-bit integer type; BYTE declares an 8-bit integer type.

The following figure is produced by the sample C language graphics application in Listing 2-1 that follows the figure. Listing 2-2 is a sample assembly language graphics program.

Figure 2-1. Output from the Sample Program

Listing 2-1. Sample Program

Listing 2-1. (continued)

Listing 2-2. Sample Assembly Language Application

Listing 2-2. (continued)

Listing 2-2. (continued)

Listing 2-2. (continued)

Listing 2-2. (continued)

Listing 2-2. (continued)

Listing 2-2. (continued)

GEM VDI Functions

The functions are grouped by type, output, and so on. Each device type requires certain functions, lists of which follow.

Opcodes

Opcodes are numbers assigned to each GEM VDI function. The device drivers recognize all opcodes, whether or not they produce any action. If an opcode is out of range, the driver performs no action.

Required Functions for Screens

Screens require the following functions and subfunctions:

Opcode	Definition
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape

id	Definition
1	Inquire addressable character cells
2	Exit alpha mode
3	Enter alpha mode
4	Cursor up
5	Cursor down
6	Cursor right
7	Cursor left
8	Home cursor
9	Erase to end of screen
10	Erase to end of line
11	Direct cursor address

- 12 Output cursor addressable text
- 15 Inquire current alpha cursor address
- 18 Place graphic cursor
- 19 Remove last graphic cursor

6 Polyline
7 Polymarker
8 Text
9 Filled area
11 Generalized Drawing Primitive (GDP)

id	Definition
1	Bar
2	Arc
3	Pie
4	Circle
5	Ellipse
6	Elliptical Arc
7	Elliptical Pie
8	Rounded rectangle
9	Filled rounded rectangle
10	Justified graphics text

12 Set character height absolute mode
14 Set color representation
15 Set polyline linetype
17 Set polyline color index
18 Set polymarker type
20 Set polymarker color index
21 Set text face
22 Set text color index
23 Set fill interior style
24 Set fill style index
25 Set fill color index
26 Inquire color representation
28 Input locator
31 Input string
32 Set writing mode
33 Set input mode
35 Inquire current polyline attributes
36 Inquire current polymarker attributes
37 Inquire current fill area attributes
38 Inquire current graphic text attributes
39 Set graphic text alignment
100 Open virtual screen workstation
101 Close virtual screen workstation
102 Extended inquire function

104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character cell height, points mode
108	Set polyline and styles
109	Copy raster, opaque
110	Transform form
111	Set mouse form
112	Set user-defined fill pattern
113	Set user-defined linestyle
114	Fill rectangle
115	Inquire input mode
116	Inquire text extent
117	Inquire character cell width
118	Exchange timer interrupt vector
121	Copy raster, transparent
122	Show cursor
123	Hide cursor
124	Sample mouse button state
125	Exchange button change vector
126	Exchange mouse movement vector
127	Exchange cursor change vector
128	Sample keyboard state information
129	Set clipping rectangle
130	Inquire face name and index
131	Inquire current face information

Required Functions for Printers

Printers require the following functions and subfunctions:

Opcode	Definition
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape

id	Definition
1	Inquire addressable character cells
20	Form advance
21	Output window
22	Clear display list
23	Output bit image file

6	Polyline
7	Polymarker
8	Text
9	Filled area
11	Generalized Drawing Primitive (GDP)

id	Definition
1	Bar
2	Arc
3	Pie
4	Circle
5	Ellipse
6	Elliptical Arc
7	Elliptical Pie
8	Rounded rectangle
9	Filled rounded rectangle
10	Justified graphics text

12	Set character height absolute mode
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
21	Set text face
22	Set text color index
23	Set fill interior style
24	Set fill style index
25	Set fill color index
26	Inquire color representation
32	Set writing mode
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes
39	Set graphic text alignment
102	Extended inquire function
104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character height points mode
108	Set polyline end styles
112	Exchange fill pattern
116	Inquire text extent
117	Inquire character cell width
129	Set clipping
130	Inquire face name and index
131	Inquire current face information

Required Functions for Plotters

Plotters require the following functions and subfunctions:

Opcode	Definition
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape

id	Definition
1	Inquire addressable character cells

Opcode	Definition
6	Polyline
7	Polymarker
8	Text
9	Filled area
11	Generalized Drawing Primitive (GDP)

id	Definition
1	Bar
2	Arc
3	Pie
4	Circle
5	Ellipse
6	Elliptical arc
7	Elliptical pie
8	Rounded rectangle
9	Filled rounded rectangle
10	Justified graphics text

Opcode	Definition
12	Set character height absolute mode
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
21	Set text face
22	Set text color index
23	Set fill interior style
24	Set fill style index
25	Set fill color index
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes

39	Set graphic text alignment
102	Extended inquire function
104	Set fill perimeter visibility
107	Set character height points mode
108	Set polyline end styles
116	Inquire text extent
117	Inquire character cell width
124	Set clipping
130	Inquire face name and index
131	Inquire current face information

Required Functions for Metafiles

Because metafiles are transportable to any device, the required functions are all those common to any device you may use. Metafiles support some inquiries by returning the op-code number. Refer to Appendix C for the metafile format of those supported inquires.

Metafiles require the following functions and subfunctions:

Opcode	Definition
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape

id	Definition
1	Inquire addressable character cells
2	Exit alpha mode
3	Enter alpha mode
20	Form advance
21	Output window
22	Clear display list
23	Output bit image file
98	Update metafile extents
99	Write metafile item
100	Change GEM VDI filename

Opcode	Definition
6	Polyline
7	Polymarker
8	Text
9	Filled area
11	Generalized Drawing Primitive (GDP)

id	Definition
1	Bar
2	Arc
3	Pie
4	Circle
5	Ellipse
6	Elliptical arc
7	Elliptical pie
8	Rounded rectangle
9	Filled rounded rectangle
10	Justified graphics text

12	Set character height absolute mode
13	Set character baseline vector
14	Set color representation
15	Set polyline linetype
16	Set polyline line width
17	Set polyline color index
18	Set polymarker type
19	Set polymarker height
20	Set polymarker color index
21	Set text face
22	Set text color index
23	Set fill interior style
24	Set fill style index
25	Set fill color index
26	Inquire color representation
32	Set writing mode
35	Inquire current polyline attributes
36	Inquire current polymarker attributes
37	Inquire current fill area attributes
38	Inquire current graphic text attributes
39	Set graphic text alignment
102	Extended inquire function
103	Contour fill
104	Set fill perimeter visibility
106	Set graphic text special effects
107	Set character height points mode
108	Set polyline end styles
112	Set fill pattern
113	Set user-defined line style pattern
114	Fill rectangle
117	Inquire character cell width
129	Set clipping rectangle
131	Inquire current face information

Available Opcodes

You can determine if a function is available in a specific driver in one of the following ways:

- o Check the information about available features returned from the Open Workstation function or the Extended Inquire function.
- o Check the selected value returned from an opcode against the requested value. If the two values are not the same, then either the function is not available or the requested value is not available, and GEM VDI selected a best fit value.

Format

The following is the format for the parameters for all GEM VDI functions.

Input Parameters

contrl(0)	--	Opcode number for the GEM VDI function.
contrl(1)	--	Number of vertices in the ptsin array. Each vertex consists of an x,y coordinate pair, so the length of the ptsin array is twice the number of specified vertices
contrl(3)	--	Length of integer array intin.
contrl(5)	--	Subfunction identification number for a Generalized Drawing Primitive (GDP) or Escape.
contrl(6)	--	Device handle.
contrl(7-n)	--	Opcode-dependent information.
intin	--	Array of integer input parameters.
ptsin	--	Array of input point coordinate data.

212

			Refer to the Extended Inquire function in Section 8 for information on how to determine the maximum size for the ptsin array.
Output Parameters	contrl(2)	--	Number of vertices in the ptsout array. Each vertex consists of an x,y coordinate pair, so the length of the ptsout array is twice the number of specified vertices.
	contrl(4)	--	Length of integer array intout.
	contrl(6)	--	Device handle.
	contrl(7-n)	--	Opcode-dependent information.
	intout	--	Array of integer output point parameters.
	ptsout	--	Array of output point coordinate data.

CALLING CONVENTIONS

Because both input and output coordinates may be converted by the GDOS, the calling routine must ensure that the vertex count, contrl(1), is set correctly. Contrl(1) must be set to 0 if no x,y coordinates are being passed to GEM VDI by the application program. In addition, the input integer count, contrl(3), must always be set. The calling routine must set contrl(3) to 0 if no integers are being passed to GEM VDI. Similarly, contrl(2), the output vertex count, and contrl(4), the output integer count, are always set correctly by GEM VDI. These values contain zeros if no information is being passed back in ptsout and intout, respectively.

The double-word addresses of the five parameter arrays are stored in a ten-word data structure referred to as a Parameter Block (PB).

Registers and Interrupts

Refer to Appendix E for the specific registers and interrupts for various operating systems.

Table 2-1. Parameter Block Contents

Address	Contents
PB	control array (contrl)
PB + 4	input parameter array (intin)
PB + 8	input point coordinate array (ptsin)
PB + 12	output parameter array (intout)
PB + 16	output point coordinate array (ptsout)

RUNNING GRAPHICS APPLICATIONS UNDER GEM VDI

To use the graphics features provided by GEM VDI, you must ensure that the following conditions are met:

1. Your application program must conform to the GEM VDI calling convention to access graphics primitives. This process involves the application making a call to the GDOS and using the interrupt for your operating system. Refer to Appendix E for the specific interrupts.

   The parameter list provides information to GEM VDI and returns information to the calling program. The details of parameter passing are in the previous section.

2. Enough stack space must be available for GEM VDI operations. This space includes a buffer area for transforming points passed to GEM VDI and some fixed overhead space. The formula to determine the required stack space is discussed under "Determining Memory Requirements" later in this section.

3. When your program is executed, the required device drivers must be present on the disk specified in the GEM VDI graphics-mode command, or in the current default drive if no drive is specified. The ASSIGN.SYS file must contain the names of your device drivers and a device ID number for each device driver. Refer to "ASSIGN.SYS" in Section 1 for information about creating an ASSIGN.SYS file.

4. After successfully compiling or assembling and linking your application program, you can run it like any program, once GEM VDI is active. You can enable GEM VDI graphics with the GEMVDI graphics-mode command, described under "Enabling Graphics" below.

ENABLING GRAPHICS

Special commands let you enable graphics functions from the command level of the operating system.

To load GEM VDI and start a non-GEM application that uses the VDI (like a test program or debugger), type the following command:

GEMVDI /FILENAME

To load GEM VDI and start a GEM application, type the following command:

GEMVDI FILENAME

To load GEM VDI and start the GEM Desktop.. application, type the following command:

GEMVDI

Each command loads GDOS and any drivers declared resident in the ASSIGN.SYS file. ASSIGN.SYS and the driver files must be located in one of the directories in the current search path.

215

Any application to be invoked by a GEMVDI command must also be located in the search path.

DISABLING GRAPHICS

When the application invoked by the GEMVDI command terminates, GEM VDI relinquishes all system memory space, leaving the maximum memory for nongraphics programs.

DETERMINING MEMORY REQUIREMENTS

To determine the amount of stack space required to run a given application, make the following calculation:

Open workstation call = approximately 128 bytes

All other calls = ptsin size + 128 bytes + the overhead requirements of the operating system

Ptsin is the point array passed to the device driver from the application program (two words for each point).

The stack requirement is the larger of the two resulting values. This stack space must be available in the application program stack area.

GEM VDI requires less than 30 kilobytes in memory for a single open driver. This space is allocated when you enter the GEM VDI graphics-mode command.

DEBUGGING GRAPHICS APPLICATIONS UNDER GEM VDI

Graphics programs can be debugged with a debugging tool. The default device drivers and GDOS are loaded after you enter the GEMVDI command. Your graphics application program is loaded in the normal manner for programs on your operating system.

End of Section 2

2/7

Section 3
CONTROL FUNCTIONS

INTRODUCTION

The control functions initialize the graphics workstation and set defaults for use with the application.

OPEN WORKSTATION

The Open Workstation function loads a graphics device driver for the application program and returns a device handle. The device is initialized with the parameters in the input array. Information about the device is returned; additional device-specific information is returned in the Extended Inquire function.

If the device is a screen, it is initialized to graphics mode. GEM VDI clears the display surface.

If the device cannot be opened, GEM VDI returns a zero as the device handle in contrl(6). Any nonzero value in contrl(6) indicates a successful operation.

Input

contrl(0)	--	Opcode = 1.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 11.
intin	--	Initial defaults (for example, linestyle, color, character size).
intin(0)	--	Device id number. This value determines which device driver to dynamically load in memory.
intin(1)	--	Linetype.
intin(2)	--	Polyline color index.
intin(3)	--	Marker type.
intin(4)	--	Polymarker color index.
intin(5)	--	Text face.
intin(6)	--	Text color index.
intin(7)	--	Fill interior style.
intin(8)	--	Fill style index.
intin(9)	--	Fill color index.

	intin(10) --	NDC to RC transformation flag. 0 = Map the full NDC space to the full RC space. 1 = Reserved. 2 = Use the RC system.
Output	contrl(2) --	Number of output vertices = 6.
	contrl(4) --	Length of intout array = 45.
	contrl(6) --	Device handle for this device.
	intout(0) --	Maximum addressable width of screen or plotter in rasters or steps, assuming a 0 start point (for example, a resolution of 640 implies an addressable area of 0-639, so intout(0)=639).
	intout(1) --	Maximum addressable height of screen or plotter in rasters or steps, assuming a 0 start point (for example, a resolution of 480 implies an addressable area of 0-479, so intout(1)=479).
	intout(2) --	Device Coordinate units flag. 0 = Device capable of producing precisely scaled image (typically a plotter or a printer). 1 = Device not capable of producing precisely scaled image (typically a film recorder).
	intout(3) --	Width of one pixel (plotter step, or aspect ratio for screen) in microns.
	intout(4) --	Height of one pixel (plotter step, or aspect ratio for screen) in microns.
	intout(5) --	Number of character heights. 0 = Continuous scaling.
	intout(6) --	Number of linetypes.
	intout(7) --	Number of line widths. 0 = Continuous scaling.

intout(8)	--	Number of marker types.
intout(9)	--	Number of marker sizes. 0 = Continuous scaling.
intout(10)	--	Number of faces supported by device (not the highest numbered face index).
intout(11)	--	Number of patterns.
intout(12)	--	Number of hatch styles.
intout(13)	--	Number of predefined colors (2 for monochrome devices). This is the number of colors that can be displayed on the device simultaneously.
intout(14)	--	Number of Generalized Drawing Primitives (GDPs).
intout(15) to intout(24)	--	Linear list of the first ten supported GDPs. The number indicates which GDP is supported. A -1 indicates the end of the list of supported GDPs. GEM VDI defines ten GDPs. 1 -- Bar 2 -- Arc 3 -- Pie slice 4 -- Circle 5 -- Ellipse 6 -- Elliptical arc 7 -- Elliptical pie 8 -- Rounded rectangle 9 -- Filled rounded rectangle 10 -- Justified graphics text
intout(25) to intout(34)	--	Linear list of attribute set associated with each GDP. 0 -- Polyline 1 -- Polymarker 2 -- Text 3 -- Fill area 4 -- None
intout(35)	--	Color capability flag.

0 -- No
1 -- Yes

intout(36) -- Text rotation capability flag.

0 -- No
1 -- Yes

intout(37) -- Fill area capability flag.

0 -- No
1 -- Yes

intout(38) -- Cell array operation capability flag.

0 -- No
1 -- Yes

intout(39) -- Number of available colors (total number of colors in color palette).

0 -- Continuous device (more than 32767 colors)
2 -- Monochrome (black and white)
>2 -- Number of colors available

intout(40) -- Number of locator devices available.

1 -- Keyboard only
2 -- Devices with keyboard and other input

intout(41) -- Number of valuator devices available.

1 -- Keyboard
2 -- If another valuator device is available

intout(42) -- Number of choice devices available.

1 -- Function keys on keyboard
2 -- If another button pad is available

intout(43) -- Number of string devices available.

1 -- Keyboard

intout(44) -- Workstation type.

0 -- Output only
1 -- Input only
2 -- Input/output
3 -- Reserved
4 -- Metafile output

ptsout(0) -- Minimum character width.
ptsout(1) -- Minimum character height in the y-axis in the current coordinate system.

The minimum and maximum character heights are the actual character body (baseline to top line), not the character extent box, which may include extra space used for interline or intercharacter spacing.

ptsout(2) -- Maximum character width.
ptsout(3) -- Maximum character height in the y-axis in the current coordinate system.
ptsout(4) -- Minimum line width in the x-axis in current coordinate system.

The minimum line width is a nominal device-dependent size. If the minimum line width used is 1 device unit, the line may not be visible on some high-resolution devices.

ptsout(5) -- 0.
ptsout(6) -- Maximum line width in the x-axis in the current coordinate system.
ptsout(7) -- 0.
ptsout(8) -- Minimum marker width in x-axis in the current coordinate system.
ptsout(9) -- Minimum marker height in x-axis in the current coordinate system.

22

ptsout(10) -- Maximum marker width in x-axis in the current coordinate system.
ptsout(11) -- Maximum marker height in x-axis in the current coordinate system.

Default Color Tables

The default color table is set up differently for monochrome and color devices.

Table 3-1. Monochrome Screens

Index	Color
0	White
1	Black

Table 3-2. Monochrome Printer/Plotters

Index	Color
0	White
1	Black

Table 3-3. Color Screens

Index	Color
0	White
1	Black
2	Red
3	Green
4	Blue
5	Cyan
6	Yellow
7	Magenta
8	White
9	Black
10	Light Red
11	Light Green
12	Light Blue
13	Light Cyan
14	Light Yellow
15	Light Magenta
16-n	Device-dependent

Other default values set by the driver during initialization are listed in Table 3-4.

Table 3-4. Default Values

Attribute	Default Value
Character height	Nominal character height
Character baseline rotation	0 degrees rotation
Text alignment	Left baseline
Text style	Normal intensity
Line width	Nominal line width
Marker height	Nominal marker height
Polyline end styles	Squared
Writing mode	Replace
Input mode	Request for all input classes (locator, valuator, choice, string)
Fill area perimeter visibility	Visible
User-defined line style	Solid
User-defined fill pattern	Solid
Cursor	Hidden
Clipping	Disabled

C BINDING

Procedure Name v_opnwk(work_in, &handle, work_out)

Data Types

```
WORD v_opnwk ( );
WORD work_in[11];
WORD handle;
WORD work_out[57];
```

Input Arguments

```
work_in[0] = intin[0]
work_in[1] = intin[1]
  .
  .
work_in[10] = intin[10]
```

Output Arguments

```
handle = contrl[6]
work_out[0] = intout[0]
work_out[1] = intout[1]
  .
  .
  .
work_out[44] = intout[44]
work_out[45] = ptsout[0]
  .
  .
  .
work_out[56] = ptsout[11]
```


225

CLOSE WORKSTATION

The Close Workstation function terminates the graphics device properly (returning you to alpha mode) and prevents any further output to the device. If the device is a screen, the alpha device is selected, and the graphics device is deselected. If the device is a printer, an update occurs if one has not just taken place. For a metafile, GEM VDI flushes the buffer and closes the metafile.

Note: Close your open virtual workstations before closing the workstation.

Input

contrl(0)	--	Opcode = 2.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.

Output

contrl(2)	--	Length of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_clswk( handle )
```

Data Types

```
WORD v_clswk ( );
WORD handle;
```

Input Arguments

```
handle = contrl[6]
```

OPEN VIRTUAL SCREEN WORKSTATION

This function allows a single physical screen to act as multiple workstations. Each workstation has access to the entire screen.

However, attribute environments for each workstation are maintained separately. For example, the workstation may have different transformation modes, clipping rectangles, and so on.

Note: Not all input devices associated with the virtual workstation will work.

The input to the Open Virtual Screen Workstation function is the device handle of a currently open physical screen workstation and an environment initialization array (see "Open Workstation"). If the virtual screen workstation can be opened, a new device handle is returned for the virtual workstation. The device capabilities arrays for the physical screen workstations are returned as they are for the Open Workstation function. If the virtual screen workstation cannot be opened, a zero is returned as the device handle to indicate an unsuccessful request.

Input

contrl(0)	--	Opcode = 100.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin = 11.
contrl(6)	--	Device handle of a previously opened screen device.
intin	--	For a description of the intin parameters required in the intin array see Open Workstation (Opcode 1).

Output	contrl(2) --	Number of output vertices = 6.
	contrl(4) --	Length of intout = 45.
	contrl(6) --	The device handle for the Virtual Screen Device just opened.
	Warning:	Contrl(6) is an input/output parameter. The value is changed to that of the Virtual Screen Workstation device handle.
	Note:	All output parameters are the same as those of Open Workstation (Opcode 1).

C BINDING

Procedure Name

```
v_opnvwk( work_in, &handle, work_out )
```

Data Types

```
WORD v_opnvwk( );
WORD handle;
WORD work_in[11];
WORD work_out[57];
```

Input Arguments

```
handle = contrl[6]
work_in[0] = intin[0]
  .
  .
work_in[10] = intin[10]
```

Output Arguments

```
work_out[0] = intout[0]
  .
  .
work_out[44] = intout[44]
work_out[45] = ptsout[0]
  .
  .
work_out[56] = ptsout[11]
```


228

CLOSE VIRTUAL SCREEN WORKSTATION	The Close Virtual Screen Workstation function terminates the virtual device and prevents any further output to it.

Input	contrl(0) --	Opcode = 101.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin = 0.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout = 0.

C BINDING

Procedure Name	v_clsvwk(handle)
Data Types	WORD v_clsvwk( ); WORD handle;
Input Arguments	handle = contrl[6]

CLEAR WORKSTATION	The Clear Workstation function erases the screen. GEM VDI sets the screen to the currently selected background color, which is defined as color index zero. If the device is a plotter without paper advance, GEM VDI prompts the operator to load a new page. If the device is a printer, data in the buffer is erased and a new page occurs. For a metafile, GEM VDI outputs the opcode. No output occurs for any device. **Note:** With GEM VDI, you do not need to do a Clear Workstation after an Open Workstation because the display is cleared at Open Workstation.

Input	contrl(0) --	Opcode = 3.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin = 0.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout = 0.

C BINDING

Procedure Name	v_clrwk(handle)
Data Types	WORD v_clrwk ( ); WORD handle;
Input Arguments	handle = contrl[6]

UPDATE WORKSTATION	The Update Workstation function causes all pending graphics commands to be executed immediately, in the order the commands were stored in the buffer. For printer drivers, you must use this function to start output to the printer. This function has no effect on screens. Plotters execute all the commands in the buffer. When the plotter buffer is empty, it returns from the Update Workstation function. For a metafile, GEM VDI outputs the opcode. **Note:** The picture is drawn to the printer but no new page occurs. A Clear Workstation causes a new page.

Input	contrl(0) --	Opcode = 4.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin = 0.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number in output vertices = 0.
	contrl(4) --	Length of intout = 0.

C BINDING

Procedure Name	v_updwk(handle)
Data Types	WORD v_updwk ( ); WORD handle;
Input Arguments	handle = contrl[6]

LOAD FONTS

This function loads the fonts associated with a particular driver in the ASSIGN.SYS file. It then makes them available to the appropriate program.

GEM VDI returns the number of newly generated font identifiers. If the fonts were already available to the workstation, no action occurs, and GEM VDI returns a zero for the number of additional font identifiers.

Note: You do not need to invoke this function if the default system fonts for a particular driver are sufficient.

Input

contrl(0) -- Opcode = 119.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Reserved for future use = 0.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of output array = 1.

intout(0) -- Number of additional font identifiers.

C BINDING

Procedure Name

additional = vst_load_fonts(handle, select)

Data Types

```
WORD vst_load_fonts( );
WORD additional;
WORD handle;
WORD select;
```

Input Arguments

handle = contrl[6]
select = intin[0]

Output Arguments

additional = intout[0]

UNLOAD FONTS

This function logically dissociates the external fonts loaded by the Load Fonts function from a device and unloads them from memory, if possible. A device handle is passed into the function identifying the device whose external fonts are to be unloaded.

If the fonts are being shared by other virtual workstations with the same root device handle, the fonts are not unloaded from memory until one of the following conditions is met:

- o all workstations that share the fonts are closed
- o all workstations that share the external fonts request that the external fonts be unloaded

The default system fonts for the workstation remain loaded and available.

Input

```
contrl(0) --  Opcode = 120.
contrl(1) --  Number of input vertices = 0.
contrl(3) --  Length of intin array = 1.
contrl(6) --  Device handle.

intin(0)  --  Reserved for future use.
```

Output

```
contrl(2) --  Number of output vertices = 0.
contrl(4) --  Length of intout array = 0.
```

C BINDING

Procedure Name	vst_unload_fonts(handle, select)
Data Types	WORD vst_unload_fonts( ); WORD handle; WORD select;
Input Arguments	handle = contrl[6] select = intin[0]

SET CLIPPING RECTANGLE

This function enables or disables clipping of all output primitives by GEM VDI. Intin(0) is a flag, which if nonzero, enables clipping. The ptsin array contains the rectangle, specified in the current coordinate system, to clip to. If intin(0) is zero, clipping is turned off. The default at Open Worksttion is for clipping to be disabled.

Input

contrl(0)	--	Opcode = 129.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Clipping flag. 0 = Turn clipping off. non-zero = Turn clipping on.
ptsin(0)	--	x-coordinate of corner of the clipping rectangle in NDC/RC units.
ptsin(1)	--	y-coordinate of corner of the clipping rectangle in NDC/RC units.
ptsin(2)	--	x-coordinate of corner diagonally across from the corner selected in ptsin(0) of the clipping rectangle in NDC/RC units.
ptsin(3)	--	y-coordinate of corner diagonally across from the corner selected in ptsin(1) of the clipping rectangle in NDC/RC units.

C BINDING

Procedure Name	vs_clip(handle, clip_flag, pxyarray)
Data Types	WORD vs_clip(); WORD handle; WORD clip_flag; WORD pxyarray[4];
Input Arguments	handle = contrl[6] clip_flag = intin[0] pxyarray[0] = ptsin[0] pxyarray[1] = ptsin[1] pxyarray[2] = ptsin[2] pxyarray[3] = ptsin[3]

End of Section 3

Section 4
OUTPUT FUNCTIONS

INTRODUCTION

The output functions display graphics primitives (polyline or circle, for example) on devices.

POLYLINE

This function displays a polyline on the graphics device. The starting point for the polyline is the first point in the input array. Lines are drawn between subsequent points in the array. GEM VDI displays a zero length line (degenerate case) as a point. GEM VDI will not display a single coordinate pair. Lines are drawn using the following current line attributes:

o color
o linetype
o line width
o end style
o current writing mode

For wide lines, the first point (ptsin(0), ptsin(1)) is drawn as shown in Figure 4-1.

Figure 4-1. First Point for Wide Lines

Input	contrl(0) --	Opcode = 6.
	contrl(1) --	Number of vertices (x,y pairs) in polyline = n. (Maximum number is returned in Extended Inquire.)
	contrl(3) --	Length of intin array = 0.
	contrl(6) --	Device handle.
	ptsin --	Array of coordinates of polyline in NDC/RC units.
	ptsin(0) --	x-coordinate of first point in NDC/RC units.
	ptsin(1) --	y-coordinate of first point in NDC/RC units.
	ptsin(2) --	x-coordinate of second point in NDC/RC units.
	ptsin(3) --	y-coordinate of second point in NDC/RC units.
		.
		.
	ptsin(2n-2) --	x-coordinate of last point in NDC/RC units.
	ptsin(2n-1) --	y-coordinate of last point in NDC/RC units.
Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name v_pline(handle, count, pxyarray)

Data Types

```
WORD v_pline ( );
WORD handle;
WORD count;
WORD pxyarray[2 * count];
```

Input Arguments

```
handle = contrl[6]
count = contrl[1]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
  .
  .
  .
pxyarray[2n-2] = ptsin[2n-2]
pxyarray[2n-1] = ptsin[2n-1]
```

POLYMARKER

This function draws markers at the points specified in the input array. GEM VDI displays the markers using the current marker attributes:

- color
- scale
- type
- writing mode

Input

contrl(0)	--	Opcode = 7.
contrl(1)	--	Number of markers = n. (Maximum number is returned in Extended Inquire.)
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
ptsin	--	Array of coordinates in NDC/RC units.
ptsin(0)	--	x-coordinate of first marker in NDC/RC units.
ptsin(1)	--	y-coordinate of first marker in NDC/RC units.
ptsin(2)	--	x-coordinate of second marker in NDC/RC units.
ptsin(3)	--	y-coordinate of second marker in NDC/RC units.
		.
		.
ptsin(2n-2)	--	x-coordinate of last marker in NDC/RC units.
ptsin(2n-1)	--	y-coordinate of last marker in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

240

C BINDING

Procedure Name v_pmarker(handle, count, pxyarray)

Data Types

```
WORD v_pmarker ( );
WORD handle;
WORD count;
WORD pxyarray[2 * count];
```

Input Arguments

```
handle = contrl[6]
count = contrl[1]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
 .
 .
 .
pxyarray[2n-2] = ptsin[2n-2]
pxyarray[2n-1] = ptsin[2n-1]
```

TEXT

This function writes graphic text to the display surface. The (x,y) position specified by the application program is the alignment point of the text string. The Set Graphic Text Alignment function establishes the relationship between the starting point of the string and the specified x,y position. The default alignment is the left baseline position of the text string. Refer to the Set Graphic Text Alignment function in Section 5 for an illustration of alignment points.

Each word of the intin array contains one character in bits 0-7. Any unsupported character is mapped to a symbol for an undefined character.

Input

contrl(0)	--	Opcode = 8.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = n.
contrl(6)	--	Device handle.
intin	--	Character string as ASCII codes in 16-bit words. The maximum number of characters equals the size of the intin array. See Extended Inquire.
ptsin(0)	--	x-coordinate of alignment point of text in NDC/RC units.
ptsin(1)	--	y-coordinate of alignment point of text in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name v_gtext(handle, x, y, string)

Data Types

```
WORD v_gtext ( );
WORD handle;
WORD x;
WORD y;
BYTE string[n];
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
string = intin
```

Note: Bytes for the string array are mapped into the eight least significant bits of intin. The string must be null-terminated.

FILLED AREA

This function fills a complex (for example, self-intersecting) polygon specified by the input array. The area is filled using the following current attributes:

- o fill area color
- o interior style (hollow, solid, pattern, hatch or user-defined)
- o writing mode
- o style index

The area is outlined with a solid line of the current fill area color if the fill area perimeter visibility is on, which is the default at Open Workstation. See the Set Fill Perimeter Visibility function in Section 5.

If a device does not have area fill capability, GEM VDI outlines the polygon using the current fill area color. The device driver ensures that the fill area is closed by connecting the first point to the last point.

GEM VDI displays a polygon with zero area as a dot. If outline isn't turned on, the degenerate case isn't displayed as a dot. GEM VDI does not display a polygon with only one endpoint. The maximum number of filled area vertices may be determined with the Extended Inquire function.

Input	contrl(0) --	Opcode = 9.
	contrl(1) --	Number of vertices in polygon = n. Maximum number returned in Extended Inquire.
	contrl(3) --	Length of intin array = 0.
	contrl(6) --	Device handle.
	ptsin --	Array of coordinates of polygon in NDC/RC units.
	ptsin(0) --	x-coordinate of first point in NDC/RC units.
	ptsin(1) --	y-coordinate of first point in NDC/RC units.
	ptsin(2) --	x-coordinate of second point in NDC/RC units.
	ptsin(3) --	y-coordinate of second point in NDC/RC units.
		. .
	ptsin(2n-2) --	x-coordinate of last point in NDC/RC units.
	ptsin(2n-1) --	y-coordinate of last point in NDC/RC units.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name v_fillarea(handle, count, pxyarray)

Data Types

```
WORD v_fillarea ( );
WORD handle;
WORD count;
WORD pxyarray[2 * count];
```

Input Arguments

```
handle = contrl[6]
count = contrl[1]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
  .
  .
  .
pxyarray[2n-2] = ptsin[2n-2]
pxyarray[2n-1] = ptsin[2n-1]
```

CELL ARRAY

With the Cell Array function, the device draws a rectangular array defined by the input parameter (x,y) coordinates and the color index array. The lower left and upper right coordinates define the extent of the rectangle. GEM VDI divides the rectangle into cells based on the number of rows and columns specified as input parameters. The color index array specifies the color for each cell.

Each cell of the rectangle is mapped to pixels on the display surface. The pixel takes the color of the cell that covers its center.

If the device does not support cell arrays, the device outlines the area with a solid line in the current line color and line width.

Note: This function is not required and may not be available on all devices.

Input

contrl(0) -- Opcode = 10.
contrl(1) -- Number of input vertices = 2.
contrl(3) -- Length of color index array.
contrl(6) -- Device handle.
contrl(7) -- Length of each row in color index array (size as declared in a high-level language).
contrl(8) -- Number of elements used in each row of color index array.
contrl(9) -- Number of rows in color index array.
contrl(10)-- Pixel operation to be performed.

(See Set Writing Mode function in Section 5 for the description of each mode.)

intin(0) -- Color index array, stored by row.

ptsin(0) -- x-coordinate of lower left corner in NDC/RC units.
ptsin(1) -- y-coordinate of lower left corner in NDC/RC units.
ptsin(2) -- x-coordinate of upper right corner in NDC/RC units.
ptsin(3) -- y-coordinate of upper right corner in NDC/RC units.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intin array = 0.

C BINDING

Procedure Name

```
v_cellarray( handle, pxyarray, row_length,
    el_used, num_rows, wrt_mode, colarray )
```

Data Types

```
WORD v_cellarray( );
WORD handle;
WORD pxyarray[4];
WORD row_length;
WORD el_used;
WORD num_rows;
WORD wrt_mode;
WORD colarray[num_rows*el_used];
```

Input Arguments

```
handle = contrl[6]
pxyarray[0] = ptsine[0]
pxyarray[1] = ptsin[1]
pxyarray[2] = ptsin[2]
pxyarray[3] = ptsin[3]
row_length = contrl[7]
el_used = contrl[8]
num_rows = contrl[9]
wrt_mode = contrl[10]
colarray[0] = intin[0]
  .
  .
colarray[n] = intin[n]
```

CONTOUR FILL

This function fills an area until it finds either the edges of the display surface or the color index stated in intin(0). This function is sometimes called a seed fill or flood fill. If intin(0) is negative, the algorithm searches for any color other than the color of the seed point. GEM VDI fills the area using the current fill area attributes.

Note: This function is not required and may not be available on all devices.

Input

contrl(0)	--	Opcode = 103.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Color index that defines the contour.
ptsin(0)	--	x-coordinate of starting point in NDC/RC units.
ptsin(1)	--	y-coordinate of starting point in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

v_contourfill(handle, x, y, index)

Data Types

```
WORD v_contourfill( );
WORD handle;
WORD x;
WORD y;
WORD index;
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
index = intin[0]
```


249

FILL RECTANGLE

This function fills a rectangular area with the pattern defined by the current fill area attributes. The rectangle is filled using all fill area attributes except outline.

Input

contrl(0) -- Opcode = 114.
contrl(1) -- Number of input vertices = 2.
contrl(3) -- Length of intin array = 0.
contrl(6) -- Device handle.

ptsin(0) -- x-coordinate of corner of destination rectangle in RC/NDC.
ptsin(1) -- y-coordinate of corner of destination rectangle in RC/NDC.
ptsin(2) -- x-coordinate of corner of destination rectangle in RC/NDC diagonally opposite corner specified in ptsin(0).
ptsin(3) -- y-coordinate of corner of destination rectangle in RC/NDC diagonally opposite corner specified in ptsin(1).

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name

```
vr_recfl( handle, pxyarray )
```

Data Types

```
WORD vr_recfl ( );
WORD handle;
WORD pxyarray[4];
```

Input Arguments

```
handle = contrl[6]
pxyarray[0] = ptsin[0]
  .
  .
  .
pxyarray[3] = ptsin[3]
```

GENERALIZED DRAWING PRIMITIVE (GDP)

The Generalized Drawing Primitive (GDP) function allows you to use the predefined primitives. The application can draw special elements, such as arcs, circles, and ellipses using this function.

The contents of the control and data arrays are different for each GDP.

For the arc, pie, elliptical arc, and elliptical pie, the information in the radius, start, and end angle variables defines the GDP.

All angle specifications are in tenths of degrees and assume that 0 degrees is 90 degrees to the right of vertical, with values increasing in the counterclockwise direction. Arcs are drawn counterclockwise. All radius specifications except for ellipse and elliptical arc, assume an extent (distance) in the x-axis. Ellipse and elliptical arc use both x and y radius values. Refer to Figure 4-2.

Figure 4-2. Angle Specification

Input

contrl(0) -- Opcode = 11.
contrl(1) -- Number of vertices in ptsin.
contrl(3) -- Length of input array intin.
contrl(5) -- Primitive id.

1 -- BAR: Uses fill area attributes (fill interior style, style index, writing mode, color and perimeter style).
4 -- CIRCLE: Uses fill area attributes (fill interior style, style index, writing mode, fill color and perimeter style).

2 -- ARC: Uses line attributes (color, linetype, writing mode, width, and end styles).

3 -- PIE: Uses fill area attributes (interior style, writing mode, fill style, fill color, and perimeter style).

5 -- ELLIPSE: Uses fill area attributes (fill interior style, writing mode, style index, color, and perimeter style).

6 -- ELLIPTICAL ARC: Uses line attributes (color, linetype, writing mode, width, and end styles).

7 -- ELLIPTICAL PIE: Uses fill area attributes (fill interior style, writing mode, style index, color and perimeter style).

8 -- ROUNDED RECTANGLE: Uses line attributes (color, linetype, writing mode, and width).

9 -- FILLED ROUNDED RECTANGLE: Uses fill area attributes (fill interior style, writing mode, style index color, and perimeter style, color, and width).

10 - JUSTIFIED GRAPHICS TEXT: Uses text attributes (face, character height, character baseline vector, color index, special effects, and alignment).

contrl(6) -- Device handle.

ptsin -- Array of coordinates for GDPs in NDC/RC units.

ptsin(0) -- x-coordinate of first point in NDC/RC units.

ptsin(1) -- y-coordinate of first point in NDC/RC units.

ptsin(2)	--	x-coordinate of second point in NDC/RC units.
ptsin(3)	--	y-coordinate of second point in NDC/RC units.
		. . .
ptsin(2n-2)	--	x-coordinate of last point in NDC/RC units.
ptsin(2n-1)	--	y-coordinate of last point in NDC/RC units.
intin	--	Angle for arcs and pies or characters for justified graphics text.

BAR

Input	contrl(0)	--	Opcode = 11.
	contrl(1)	--	Number of input vertices = 2.
	contrl(3)	--	Length of intin array = 0.
	contrl(5)	--	Primitive id = 1.
	contrl(6)	--	Device handle.
	ptsin(0)	--	x-coordinate of corner of bar in NDC/RC units.
	ptsin(1)	--	y-coordinate of corner of bar in NDC/RC units.
	ptsin(2)	--	x-coordinate of corner diagonally opposite the corner selected in ptsin(0) of bar in NDC/RC units.
	ptsin(3)	--	y-coordinate of corner diagonally opposite the corner selected in ptsin(1) of bar in NDC/RC units.

Output	contrl(2)	--	Number of output vertices = 0.
	contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_bar( handle, pxyarray )
```

Data Types

```
WORD v_bar ( );
WORD handle;
WORD pxyarray[4];
```

Input Arguments

```
handle = contrl[6]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
pxyarray[2] = ptsin[2]
pxyarray[3] = ptsin[3]
```

ARC & PIE	These functions are not required and may not be available on all devices. GEM VDI requires the specification of the arc by the angle (intin(0),intin(1)).

Input	contrl(0)	--	Opcode = 11.
	contrl(1)	--	Number of input vertices = 4.
	contrl(3)	--	Length of intin array = 2.
	contrl(5)	--	Primitive id. 2 = ARC 3 = PIE
	contrl(6)	--	Device handle.
	intin(0)	--	Start angle (in tenths of degrees 0-3600), counterclockwise.
	intin(1)	--	End angle (in tenths of degrees 0-3600).
	ptsin(0)	--	x-coordinate of center point of arc in NDC/RC units.
	ptsin(1)	--	y-coordinate of center point of arc in NDC/RC units.
	ptsin(2)	--	0.
	ptsin(3)	--	0.
	ptsin(4)	--	0.
	ptsin(5)	--	0.
	ptsin(6)	--	Radius in x-coordinate NDC/RC units.
	ptsin(7)	--	0.

Output	contrl(2)	--	Number of output vertices = 0.
	contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_arc( handle, x, y, radius, begang, endang )
v_pieslice( handle, x, y, radius, begang,
endang )
```

Data Types

```
WORD v_arc ( );
WORD handle;
WORD x, y;
WORD radius;
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
radius = ptsin[6]
begang = intin[0]
endang = intin[1]
```

CIRCLE This function is not required and may not be supported on all devices.

Input

contrl(0) -- Opcode = 11.
contrl(1) -- Number of input vertices = 3.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Primitive id = 4.
contrl(6) -- Device handle.

ptsin(0) -- x-coordinate of center point of circle in NDC/RC units.
ptsin(1) -- y-coordinate of center point of circle in NDC/RC units.
ptsin(2) -- 0.
ptsin(3) -- 0.
ptsin(4) -- Radius in x-coordinate NDC/RC units.
ptsin(5) -- 0.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name v_circle(handle, x, y, radius)

Data Types

```
WORD v_circle ( );
WORD handle;
WORD x, y;
WORD radius;
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
radius = ptsin[4]
```

ELLIPTICAL ARC AND PIE

Input

contrl(0)	--	Opcode = 11.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 2.
contrl(5)	--	Primitive id. 6 = ELLIPTICAL ARC 7 = ELLIPTICAL PIE SLICE
contrl(6)	--	Device handle.
intin(0)	--	Start angle (in tenths of degrees 0-3600), counterclockwise.
intin(1)	--	End angle (in tenths of degrees 0-3600).
ptsin(0)	--	x-coordinate of center point of arc in NDC/RC units.
ptsin(1)	--	y-coordinate of center point of arc in NDC/RC units.
ptsin(2)	--	Radius of X-axis in NDC/RC units.
ptsin(3)	--	Radius of Y-axis in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

95-8

C BINDING

Procedure Name

```
v_ellarc( handle, x, y, xradius, yradius,
  begang, endang )

v_ellpie( handle, x, y, xradius, yradius,
  begang, endang )
```

Data Types

```
WORD v_ellarc ( );
WORD v_ellpie ( );
WORD handle;
WORD x, y;
WORD xradius;
WORD yradius;
WORD begang;
WORD endang;
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
xradius = ptsin[2]
yradius = ptsin[3]
begang = intin[0]
endang = intin[1]
```

ELLIPSE

Input

contrl(0)	--	Opcode = 11.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 0.
contrl(5)	--	Primitive id = 5.
contrl(6)	--	Device handle.
ptsin(0)	--	x-coordinate of center point of ellipse in NDC/RC units.
ptsin(1)	--	y-coordinate of center point of ellipse in NDC/RC units.
ptsin(2)	--	Radius of X-axis in NDC/RC units.
ptsin(3)	--	Radius of Y-axis in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_ellipse( handle, x, y, xradius, yradius )
```

Data Types

```
WORD v_ellipse ( );
WORD handle;
WORD x, y;
WORD xradius;
WORD yradius;
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
xradius = ptsin[2]
yradius = ptsin[3]
```

ROUNDED AND FILLED ROUNDED RECTANGLE

A rectangle with rounded corners is output to the workstation. The rectangle is defined by specifying its lower left and upper right corners.

The Rounded Rectangle GDP assumes the attributes of a polyline primitive. The Filled Rounded Rectangle GDP assumes the attributes of a filled area primitive.

Input

contrl(0) -- Opcode = 11.
contrl(1) -- Number of input vertices = 2.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Primitive id.

8 = Rounded Rectangle
9 = Filled Rounded Rectangle

contrl(6) -- Device handle.
ptsin(0) -- x-coordinate of corner of rectangle in NDC/RC units.
ptsin(1) -- y-coordinate of corner of rectangle in NDC/RC units.
ptsin(2) -- x-coordinate of corner diagonally opposite corner selected in ptsin(0) of rectangle in NDC/RC units.
ptsin(3) -- y-coordinate of corner diagonally opposite corner selected in ptsin(1) of rectangle in NDC/RC units.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

261

C BINDING

Procedure Name

```
v_rbox( handle, xyarray )
v_rfbox( handle, xyarray)
```

Data Types

```
WORD ( v_rbox );
WORD ( v_rfbox );
WORD handle;
WORD xyarray[4];
```

Input Arguments

```
handle = contrl[6];
attributes = intin[0];
xyarray[0] = ptsin[0];
xyarray[1] = ptsin[1];
xyarray[2] = ptsin[2];
xyarray[3] = ptsin[3];
```


262

JUSTIFIED GRAPHICS TEXT	This function outputs graphics text to the workstation display surface and attempts to perform both left and right justification. The text string is aligned at the requested string alignment points passed in, using the current text alignment attributes. Extra spacing may be inserted or deleted by the driver between words or characters (or both) so that the string will have the requested length. Either form of spacing modification (inter-character or inter-word) can be suppressed by so specifying in the provided parameter.

Input

contrl(0)	--	Opcode = 11.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 2 + n (characters in string).
contrl(5)	--	Primitive id = 10.
contrl(6)	--	Device handle.
intin(0)	--	Inter-word spacing flag. 0 = Doesn't allow GEM VDI to modify inter-word spacing. nonzero = Allows GEM VDI to modify inter-word spacing.
intin(1)	--	Inter-character spacing flag. 0 = Doesn't allow GEM VDI to modify inter-character spacing. nonzero = Allows GEM VDI to modify inter-character spacing.
intin(2)	--	First character of text string. . . .
intin(n+1)	--	Last character of text string.

263

	ptsin(0) --	x-coordinate of the text alignment point, in NDC/RC units.
	ptsin(1) --	y-coordinate of the text alignment point, in NDC/RC units.
	ptsin(2) --	Requested length of the string, in x-axis NDC/RC units.
	ptsin(3) --	0.
Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name

```
v_justified(handle, x, y, string, length,
  word_space, char_space);
```

Data Types

```
WORD v_justified();
WORD handle;
WORD x, y;
WORD length;
WORD word_space;
WORD char_space;
BYTE string[];
```

Input Arguments

```
handle = contrl[6];
x = ptsin[0];
y = ptsin[1];
length = ptsin[2];
word_space = intin[0];
char_space = intin[1];
string[j] = intin[j+2];
```

Note: Bytes for the string array are mapped into the eight least significant bits of intin words.

Note: The string array must be null-terminated.

End of Section 4

215

Section 5
ATTRIBUTE FUNCTIONS

INTRODUCTION

Attribute functions determine qualities of all subsequent output primitives such as color, type, style, and height.

SET WRITING MODE

This function selects the writing mode used for subsequent drawing operations. The writing mode specifies the operation performed between the color indices of the current pixel (source) and the existing pixel (destination), thus affecting the way new pixels from lines, markers, filled areas, and text are placed on the display. Four modes exist: replace, transparent, XOR, and reverse transparent. If the requested writing mode is out of range, GEM VDI selects replace mode, 1.

Table 5-1 lists the writing modes and their numerical assignments.

Table 5-1. Writing Modes

Number	Mode
1	Replace
2	Transparent
3	XOR
4	Reverse Transparent

For the Boolean expressions of the modes given below, the definitions in Table 5-2 apply.

Table 5-2. Terms

Term	Definition
mask	line style or fill pattern
fore	selected color after mapping from GEM VDI
back	color 0 after mapping from GEM VDI (white is default)
old	current color value
new	replacement color value

Replace

Replace mode is insensitive to the currently displayed image. Any information already displayed is replaced. The following is the Boolean expression for replace mode:

new = (fore AND mask) OR (back AND NOT mask)

Transparent

Transparent mode only affects the pixels where the mask is 1. These are changed to the fore value. The following is the Boolean expression for transparent mode:

new = (fore AND mask) OR (old AND NOT mask)

XOR

XOR mode reverses the bits representing the color. The following is the Boolean expression for XOR mode:

new = mask XOR old

Reverse Transparent	Reverse transparent mode only affects the pixels where the mask is 0. These are changed to the fore value. The following is the Boolean expression for reverse transparent mode: new = (old AND mask) OR (fore AND NOT mask)
Input	contrl(0) -- Opcode = 32. contrl(1) -- Number of input vertices = 0. contrl(3) -- Length of intin array = 1. contrl(6) -- Device handle. intin(0) -- Writing mode requested.
Output	contrl(2) -- Number of output vertices = 0. contrl(4) -- Length of intout array = 1. intout(0) -- Writing mode selected.

C BINDING

Procedure Name	set_mode = vswr_mode(handle, mode)
Data Types	WORD set_mode; WORD vswr_mode ( ); WORD handle; WORD mode;
Input Arguments	handle = contrl[6] mode = intin[0]
Output Arguments	set_mode = intout[0]

SET COLOR REPRESENTATION

This function associates a color index with the color specified in RGB (Red, Green, Blue) units. On a monochrome device, GEM VDI maps any percentage of color to white. GEM VDI maps any color intensity of a value less than 0 to 0 and greater than 1000 to 1000. If the application requests a color index that is out of range, GEM VDI performs no operation. GEM VDI references the background color as color index zero.

Note: If no color lookup table exists, GEM VDI performs no operation with this function. The Extended Inquire function returns the availability of the lookup table.

Input

contrl(0)	--	Opcode = 14.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 4.
contrl(6)	--	Device handle.
intin(0)	--	Color index.
intin(1)	--	Red color intensity (in tenths of percent, 0-1000).
intin(2)	--	Green color intensity.
intin(3)	--	Blue color intensity.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name vs_color(handle, index, rgb_in)

Data Types

```
WORD vs_color ( );
WORD handle;
WORD index;
WORD rgb_in[3];
```

Input Arguments

```
handle = contrl[6]
index = intin[0]
rgb_in[0] = intin[1]
rgb_in[1] = intin[2]
rgb_in[2] = intin[3]
```

SET POLYLINE LINE TYPE

This function sets the line type for subsequent polyline operations. The total number of line styles available is device-dependent, but all devices support at least six. If the requested line style is out of range, GEM VDI selects solid (1) line style. The pixel value in the pattern word is 1 = pixel on (active); 0 = pixel off.

Style		16 Bits MSB LSB
1	solid	1111111111111111
2	long dash	1111111111110000
3	dot	1110000011100000
4	dash,dot	1111111000111000
5	dash	1111111100000000
6	dash,dot,dot	1111000110011000
7	user-defined style	16 bits (1 word) Most Significant Bit = first pixel displayed.
8-n	device-dependent	

Line style seven, user-defined style, uses the pattern the Set User-defined Line Style Pattern function defines. This pattern defaults to solid until the user defines it.

Note: If a nondefault line width is used, the device may draw the thickened line using a solid line style and may change the writing mode.

271

Input	contrl(0) --	Opcode = 15.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 1.
	contrl(6) --	Device handle.
	intin(0) --	Requested line style.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 1.
	intout(0) --	Line style selected.

C BINDING

Procedure Name set_type = vsl_type(handle, style)

Data Types

```
WORD set_type ;
WORD vsl_type ( );
WORD handle;
WORD style;
```

Input Arguments

```
handle = contrl[6]
style = intin[0]
```

Output Arguments set_type = intout[0]

SET USER-DEFINED LINE STYLE PATTERN

This function sets the current user-defined line style pattern word in the device driver to the value in the specified 16-bit pattern word.

The Most Significant Bit (MSB) of the pattern word is the first pixel in the line. This line style is used for subsequent polyline operations when the application selects user-defined line style, index 7.

Input

contrl(0) -- Opcode = 113.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Line style pattern word, 16 bits.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name

```
vsl_udsty( handle, pattern )
```

Data Types

```
WORD vsl_udsty ( );
WORD handle;
WORD pattern;
```

Input Arguments

```
handle = contrl[6]
pattern = intin[0]
```

SET POLYLINE LINE WIDTH

This function sets the width of lines for subsequent polyline operations. The available line width closest to but not greater than the requested line width is used. Line widths are odd numbers that begin at three. If you select two in Raster Coordinates, GEM VDI returns one, which is a line one pixel wide.

Note: This function is not required and may not be available on all devices. Thickened lines may be rendered on the device using solid line type, rather than a requested line type.

Input

contrl(0)	--	Opcode = 16.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
ptsin(0)	--	Requested line width in x-axis in NDC/RC units.
ptsin(1)	--	0.

Output

contrl(2)	--	Number of output vertices = 1.
contrl(4)	--	Length of intout array = 0.
ptsout(0)	--	Selected line width in x-axis of the NDC/RC units.
ptsout(1)	--	0.

C BINDING

Procedure Name	set_width = vsl_width(handle, width)
Data Types	WORD set_width; WORD vsl_width; WORD handle; WORD width;
Input Arguments	handle = contrl[6] width = ptsin[0]
Output Arguments	set_width = ptsout[0]

SET POLYLINE COLOR INDEX

This function sets the color index for subsequent polyline operations. The Set Color Representation function determines the color the index represents. At least two color indices, 0 and 1, are supported (monochrome). Color indices range from 0 to a device-dependent maximum. If the application requests an index that is out of range, GEM VDI selects color index 1.

Input

contrl(0) -- Opcode = 17.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Requested color index.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Color index selected.

C BINDING

Procedure Name

```
set_color = vsl_color( handle, color_index )
```

Data Types

```
WORD set_color;
WORD vsl_color ( );
WORD handle;
WORD color_index;
```

Input Arguments

```
handle = contrl[6]
color_index = intin[0]
```

Output Arguments

```
set_color = intout[0]
```


276

SET POLYLINE END STYLES

This function sets the style for the ends of a polyline. The style may be any of the following:

0 - squared (default)
1 - arrow
2 - rounded

The two ends of a polyline may have different styles. If an invalid style is requested, a squared end style (0) is used.

Both the squared style and the arrow style end at the end of the polyline. The rounded style is drawn such that the center of the rounding is at the end of the polyline.

Input

contrl(0) -- Opcode = 108.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 2.
contrl(6) -- Device handle.

intin(0) -- End style for beginning point of polyline.

0 - squared (default)
1 - arrow
2 - rounded

intin(1) -- End style for ending point of polyline.

0 - squared (default)
1 - arrow
2 - rounded

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name	vsl_ends(handle, beg_style, end_style)
Data Types	WORD vsl_ends(); WORD handle; WORD beg_style; WORD end_style;

Input Arguments	handle = contrl[6];
	beg_style = intin[0];
	end_style = intin[1];

SET POLYMARKER TYPE

This function sets the marker type for subsequent polymarker functions. The total number of markers available is device-dependent, but GEM VDI always defines at least six marker types:

1 - .	Dot
2 - +	Plus
3 - *	Asterisk
4 - O	Square
5 - X	Diagonal Cross
6 - <>	Diamond
7 ... n	Device-dependent

If the requested marker type is out of range, GEM VDI uses an asterisk, type 3. Marker 1 is the smallest dot GEM VDI displays on the device; it cannot be scaled.

Input

contrl(0)	--	Opcode = 18.
contrl(1)	--	Numbers of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Requested polymarker type.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Polymarker type selected.

C BINDING

Procedure Name	set_type = vsm_type(handle, symbol)
Data Types	WORD set_type; WORD vsm_type ( ); WORD handle; WORD symbol;
Input Arguments	handle = contrl[6] symbol = intin[0]
Output Arguments	set_type = intout[0]

SET POLYMARKER HEIGHT

This function sets a polymarker height for subsequent polymarker functions. If the selected height does not exist, GEM VDI selects the next smaller height. The driver returns the actual height selected in the ptsout array.

Input

contrl(0)	--	Opcode = 19.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
ptsin(0)	--	0.
ptsin(1)	--	Requested polymarker height in y-axis in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 1.
contrl(4)	--	Length of intout array = 0.
ptsout(0)	--	Polymarker width selected in x-axis in NDC/RC units.
ptsout(1)	--	Polymarker height selected in y-axis in NDC/RC units.

C BINDING

Procedure Name

```
set_height = vsm_height( handle, height )
```

Data Types

```
WORD set_height;
WORD vsm_height ( );
WORD handle;
WORD height;
```

Input Arguments

```
handle = contrl[6]
height = ptsin[1]
```

Output Arguments

```
set_height = ptsout[1]
```

SET POLYMARKER COLOR INDEX	This function sets the color index for subsequent polymarker functions. The Set Color Representation function specifies the value of the index. At least two color indices are always supported (monochrome). If the index is out of range, GEM VDI selects color index 1.

Input	contrl(0) --	Opcode = 20.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 1.
	contrl(6) --	Device handle.
	intin(0) --	Requested polymarker color index.

Output	contrl(2) --	number of output vertices = 0.
	contrl(4) --	length of intout array = 1.
	intout(0) --	Polymarker color index selected.

C BINDING

Procedure Name set_color = vsm_color(handle, color_index)

Data Types

```
WORD set_color;
WORD vsm_color ( );
WORD handle;
WORD color_index;
```

Input Arguments

```
handle = contrl[6]
color_index = intin[0]
```

Output Arguments

```
set_color = intout[0]
```

SET CHARACTER HEIGHT, ABSOLUTE MODE

This function sets the current graphic text character height in NDC/RC units. The specified height is the distance from the character baseline to the top of the character cell, rather than the character cell height.

GEM VDI returns the selected height and width information to the application. GEM VDI returns both the distance from the baseline to top line selected and the size of a character cell. (See Figure 5-1 under "Set Character Height, Points Mode.") For fixed (monospaced) faces GEM VDI returns the width of a character and the width of a character cell. For proportional faces, GEM VDI returns the width of the widest character and the width of the widest character cell in the face.

If the desired character height does not map exactly to a device size, GEM VDI selects the closest character size that does not exceed the requested size.

Input

contrl(0)	--	Opcode = 12.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
ptsin(0)	--	0.
ptsin(1)	--	Requested character height in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 2.
contrl(4)	--	Length of intout array = 0.
ptsout(0)	--	Character width selected in NDC/RC units.
ptsout(1)	--	Character height selected in NDC/RC units.
ptsout(2)	--	Character cell width in NDC/RC units.
ptsout(3)	--	Character cell height in NDC/RC units.

C BINDING

Procedure Name

```
vst_height( handle, height, &char_width,
   &char_height, &cell_width, &cell_height )
```

Data Types

```
WORD vst_height ( );
WORD handle;
WORD height;
WORD char_width;
WORD char_height;
WORD cell_width;
WORD cell_height;
```

Input Arguments

```
handle = contrl[6]
height = ptsin[1]
```

Output Arguments

```
char_width = ptsout[0]
char_height = ptsout[1]
cell_width = ptsout[2]
cell_height = ptsout[3]
```


284

SET CHARACTER CELL HEIGHT, POINTS MODE

This function sets the current graphic text character height in printer points. A point is 1/72 of an inch. The specified height is the distance between the baseline of one line of text and the baseline of the next line of text, which is the character cell height.

The driver returns the selected point size of the character. Height and width information is returned in NDC/RC units. GEM VDI returns the character height, character width, cell height, and the cell width, as shown in Figure 5-1. For proportional faces, GEM VDI returns the width of the widest character and the widest character cell in the face.

If the desired character height does not map exactly to a device size, GEM VDI selects the closest character size not exceeding the requested size.

Figure 5-1. Character Cell Definition

Input

contrl(0)	--	Opcode = 107.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Cell height in points.

Output	contrl(2) --	Number of output vertices = 2.
	contrl(4) --	Length of intout array = 1.
	intout(0) --	Selected cell height in points.
	ptsout(0) --	Character width selected in NDC/RC units.
	ptsout(1) --	Character height selected in NDC/RC units.
	ptsout(2) --	Character cell width in NDC/RC units.
	ptsout(3) --	Character cell height in NDC/RC units.

C BINDING

Procedure Name

```
set_point = vst_point( handle, point,
   &char_width, &char_height, &cell_width,
   &cell_height )
```

Data Types

```
WORD set_point;
WORD vst_point( );
WORD handle;
WORD point;
WORD char_width;
WORD char_height;
WORD cell_width;
WORD cell_height;
```

Input Arguments

```
handle = contrl[6]
point = intin[0]
```

Output Arguments

```
set_point( ) = intout[0]
char_width = ptsout[0]
char_height = ptsout[1]
cell_width = ptsout[2]
cell_height = ptsout[3]
```

SET CHARACTER BASELINE VECTOR

This function requests an angle of rotation specified in tenths of degrees for the character baseline vector, which specifies the baseline for subsequent graphic text. The driver returns the selected baseline vector to the application. The selected baseline vector is a best-fit match to the requested value.

See Figure 5-2 for a depiction of how angles are specified to GEM VDI.

Figure 5-2. Angle Specification

Note: This function is not required and may not be supported on all devices. The Extended Inquire function returns the availability of this function.

Input

contrl(0)	--	Opcode = 13.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Requested angle of rotation of character baseline (in tenths of degrees, 0 - 3600).

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Angle of rotation of character baseline selected (in tenths of degrees 0-3600).

C BINDING

Procedure Name	set_baseline = vst_rotation(handle, angle)
Data Types	WORD set_baseline; WORD vst_rotation ( ); WORD handle; WORD angle;
Input Arguments	handle = contrl[6] angle = intin[0]
Output Arguments	set_baseline = intout[0]

SET TEXT FACE

This function selects a graphic character face for subsequent graphic text operations. Face 1 is a built-in face. The other faces are external and may be loaded with the Load Face function. Some faces may not be supported on all devices. Face names and indices may be determined by using Inquire Face Name.

Input

contrl(0)	--	Opcode = 21.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Requested software text face number.

1 - System face
2 - Swiss 721
3 - Swiss 721 Thin
4 - Swiss 721 Thin Italic
5 - Swiss 721 Light
6 - Swiss 721 Light Italic
7 - Swiss 721 Italic
8 - Swiss 721 Bold
9 - Swiss 721 Bold Italic
10 - Swiss 721 Heavy
11 - Swiss 721 Heavy Italic
12 - Swiss 721 Black
13 - Swiss 721 Black Italic
14 - Dutch 801 Roman
15 - Dutch 801 Italic
16 - Dutch 801 Bold
17 - Dutch 801 Bold Italic

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Text face selected.

C BINDING

Procedure Name	set_font = vst_font(handle, font)
Data Types	WORD set_font; WORD vst_font ( ); WORD handle; WORD font;
Input Arguments	handle = contrl[6] font = intin[0]
Output Arguments	set_font = intout[0]

SET GRAPHIC TEXT COLOR INDEX

This function sets the color index for subsequent graphic text operations. The Set Color Representation function determines the color represented by the color index. All devices support at least two color indices, 0 and 1 (monochrome). Color indices range from 0 to a device-dependent maximum. If the requested index is out of range, GEM VDI selects color index 1.

Input

contrl(0) -- Opcode = 22.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Requested text color index.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Text color index selected.

C BINDING

Procedure Name

```
set_color = vst_color( handle, color_index )
```

Data Types

```
WORD set_color;
WORD vst_color ( );
WORD handle;
WORD color_index;
```

Input Arguments

```
handle = contrl[6]
color_index = intin[0]
```

Output Arguments

```
set_color = intout[0]
```

SET GRAPHIC TEXT SPECIAL EFFECTS

This function sets text special effects for subsequently displayed graphic text. The following effects are available:

- o thickened
- o light intensity
- o skewed
- o underlined
- o outlined
- o shadowed
- o any combination of the above

GEM VDI treats the integer in intin(0) as a bit pattern. The attributes set correspond to the setting in the six least significant bits.

Table 5-3. Attribute Bit Mapping

Bit	Value	Description
0		Thickened
	0	thickened not selected
	1	set style to thickened
1		Intensity
	0	normal intensity
	1	light intensity
2		Skewed
	0	skewed not selected
	1	set style to skewed
3		Underlined
	0	do not underline
	1	text is underlined
4		Outline
	0	no outline
	1	outline
5		Shadow
	0	no shadow
	1	shadow

292

For example, if intin(0) = 9 (1001 binary), the text style is set to thickened and underlined.

For effects not supported on a device, GEM VDI returns those bits set to 0.

Input

contrl(0) -- Opcode = 106.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Special effect word.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Styles actually selected (style word with the appropriate bits set).

Figure 5-3. Graphic Text Special Effects

C BINDING

Procedure Name

```
set_effect = vst_effects( handle, effect )
```

Data Types

```
WORD set_effect;
WORD vst_effects( );
WORD handle;
WORD effect;
```

Input Arguments

```
handle = contrl[6]
effect = intin[0]
```

Output Arguments

```
set_effect = intout[0]
```


294

SET GRAPHIC TEXT ALIGNMENT

This function sets horizontal and vertical alignment for graphic text. Horizontal means in the direction of the baseline; vertical is perpendicular to the baseline. This function controls the positioning of the text string in relation to the graphic text position. The default alignment places the left baseline corner of the string at the graphic text position.

If the application requests an invalid horizontal alignment, GEM VDI selects the default, left. If the application requests an invalid vertical alignment, GEM VDI selects the default, baseline.

Figure 5-4. Graphic Text Alignment

Input

contrl(0)	--	Opcode = 39.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 2.
contrl(6)	--	Device handle.
intin(0)	--	Horizontal alignment requested 0 = left justified (default) 1 = center justified 2 = right justified
intin(1)	--	Vertical alignment requested 0 = baseline (default) 1 = half line 2 = ascent line 3 = bottom 4 = descent 5 = top

---

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 2.
intout(0)	--	Horizontal alignment selected.
intout(1)	--	Vertical alignment selected.

C BINDING

Procedure Name

```
vst_alignment( handle, hor_in, vert_in,
  &hor_out, &vert_out )
```

Data Types

```
WORD vst_alignment( );
WORD handle;
WORD hor_in;
WORD vert_in;
WORD hor_out;
WORD vert_out;
```

Input Arguments

```
handle = contrl[6]
hor_in = intin[0]
vert_in = intin[1]
```

Output Arguments

```
hor_out = intout[0]
vert_out = intout[1]
```


296

SET FILL INTERIOR STYLE

This function sets the fill interior style used in subsequent polygon fill operations. If the application requests an unavailable style, the area is hollow filled. GEM VDI returns the selected style to the application. Hollow style fills the interior with the current background color(index 0). Solid style fills the area with the currently selected fill color.

Input

contrl(0) -- Opcode = 23.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Requested fill interior style.

0 - hollow
1 - solid
2 - pattern
3 - hatch
4 - user-defined style

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Fill interior style selected.

C BINDING

Procedure Name set_interior = vsf_interior(handle, style)

Data Types

```
WORD set_interior;
WORD vsf_interior ( );
WORD handle;
WORD style;
```

Input Arguments

handle = contrl[6]
style = intin[0]

Output Arguments	set_interior = intout[0]

SET FILL STYLE INDEX

This function selects a fill style based on the fill interior style. This index has no effect if the interior style is hollow, solid, or user-defined. Indices range from 1 to a device-dependent maximum. If the requested index is not available, GEM VDI uses index style 1. The index references a hatch style if the selected fill interior style is hatch, or a pattern if the selected interior fill style is pattern.

Figure 5-5 shows the available fill styles. Under each rectangle in Figure 5-5 are two numbers, separated by a comma. The number to the left of the comma corresponds to the style: Hollow, Pattern, or Hatch. The number to the right of the comma corresponds to the index for the particular pattern or hatch.

Figure 5-5. Fill Styles and Indices

Note: 1,n (i.e., Style 1, followed by any index) produces the same result as 2,8.

For patterns, index 1 maps to the lowest intensity pattern on the device. The pattern is always monochrome and uses the current fill area color for foreground pixels.

Input	contrl(0)	--	Opcode = 24.
	contrl(1)	--	Number of input vertices = 0.
	contrl(3)	--	Length of intin array = 1.
	contrl(6)	--	Device handle.
	intin(0)	--	Requested fill style index for pattern or hatch fill.

Output	contrl(2)	--	Number of output vertices = 0.
	contrl(4)	--	Length of intout array = 1.
	intout(0)	--	Fill style index selected for pattern or hatch fill.

C BINDING

Procedure Name

```
set_style = vsf_style( handle, style_index )
```

Data Types

```
WORD set_style;
WORD vsf_style ( );
WORD handle;
WORD style_index;
```

Input Arguments

```
handle = contrl[6]
style_index = intin[0]
```

Output Arguments

```
set_style = intout[0]
```

SET FILL COLOR INDEX

This function sets the color index for subsequent polygon fill functions. The Set Color Representation function determines the color represented by the color index. All devices support at least two color indices, 0 and 1 (monochrome). Color indices range from 0 to a device-dependent maximum. If the requested index is out of range, GEM VDI selects color index 1.

Input

contrl(0)	--	Opcode = 25.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Requested fill color index.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Fill color index selected.

C BINDING

Procedure Name

```
set_color = vsf_color( handle, color_index )
```

Data Types

```
WORD set_color;
WORD vsf_color ( );
WORD handle;
WORD color_index;
```

Input Arguments

```
handle = contrl[6]
color_index = intin[0]
```

Output Arguments

```
set_color = intout[0]
```


300

SET FILL PERIMETER VISIBILITY

This function turns the outline of a fill area on or off. When visibility is on (the default at Open Workstation) the border of a fill area is drawn in the current fill area color with a solid line. When visibility is off, no outline is drawn. Any nonzero value of the visibility flag causes the perimeter to be visible.

Input

contrl(0)	--	Opcode = 104.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Visibility flag. zero - invisible nonzero - visible

---

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Visibility selected.

C BINDING

Procedure Name

```
set_perimeter = vsf_perimeter( handle, per_vis )
```

Data Types

```
WORD set_perimeter;
WORD vsf_perimeter ( );
WORD handle;
WORD per_vis;
```

Input Arguments

```
handle = contrl[6]
per_vis = intin[0]
```

Output Arguments

```
set_perimeter = intout[0]
```

SET USER-DEFINED FILL PATTERN

This function redefines the user-definable fill pattern.

For the pattern data, bit 15 of word 1 is the upper left bit of the pattern. Bit 0 of word 16 is the lower right bit of the pattern. Bit zero is the Least Significant Bit of the word. Words are stored in the same format as 16-bit integers

For a single plane pattern, a bit value of 1 indicates foreground color. A bit value of 0 indicates the background color. The color used for the foreground is determined by the current fill area color index.

For a multiple plane pattern, the number of full 16-by-16 planes defined are used in the fill operation: planes = contrl(3) / 16. Any unspecified planes are zeroed. Note that the writing mode must be set to replace (mode 1), when using a multiplane fill pattern.

The defined pattern is referenced by the Set Fill Interior Style function as style 4 and by the Fill Rectangle function.

Input

contrl(0)	--	Opcode = 112.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 16 to n.
contrl(6)	--	Device handle.
intin(0) to intin(15)	--	First plane of fill pattern.
intin(16) to intin(29)	--	Second plane of fill pattern.
		.
		.
		.
intin(n-15) to intin(n)	--	Last plane of fill pattern.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name	vsf_udpat(handle, pfill_pat, planes)
Data Types	WORD vsf_udpat; WORD handle; WORD pfill_pat[16 x n where n > 0] WORD planes;
Input Arguments	handle = contrl[6] pfill_ . . . pfill_pat planes = contrl[3]/16

End of Section 5

Section 6
RASTER OPERATIONS

INTRODUCTION

Raster operations perform logic operations on rectangular blocks of bits in memory and on rectangular blocks of pixels on physical devices.

MEMORY FORM DEFINITION BLOCK

A raster area is defined by a Memory Form Definition Block (MFDB). An MFDB consists of the following components:

- o A 32-bit pointer to the memory address of the upper left corner of the first plane of the raster area. This pointer corresponds to an offset-segment pointer for 8086-based microcomputers. If all 32 bits of this pointer are 0, the MFDB is for a physical device, and the other parameters are ignored.
- o The height and width of the raster area in pixels.
- o The width of the raster area in words. This value is equal to the width of the raster area in pixels, divided by the word size.
- o The number of planes in the raster area.
- o A flag indicating whether the format of the raster area is standard or device-dependent.
- o Some locations reserved for future use.

A raster area must start on a word boundary and have a width that is an integral multiple of the word size.

Figure 6-1. Memory Form Definition Block

RASTER AREA FORMATS

Two memory formats are associated with raster areas:

- o device-specific format
- o well-defined standard format

GEM VDI provides a function to transform a raster area from one format to another. You must transform a form before using Copy Raster.

The form format flag can have two values:

0 - The form is in device-specific format.
1 - The form is in standard format.

The layout of a standard form format is as follows (see also Figure 6-2):

- Plane based - The planes are contiguous blocks of memory, each having the same x,y resolution. A monochrome implementation has a single plane. A color index is mapped to a pixel value with each plane representing one bit in the value. Tables 6-1 and 6-2 define the pixel-value-to-color-index mapping for eight-color and sixteen-color screens, respectively.

- Most Significant Bit in a word (16-bit integer) is the leftmost bit in the image. Note that the data is stored in the same format as 16-bit integers.

- Words are arranged sequentially along a row with the first word being on the left edge of the row.

Table 6-1. Pixel Value to Color Index Mapping for 8-color Screens

Pixel Value	Color Index	Color
000	0	white
001	2	red
010	3	green
011	6	yellow
100	4	blue
101	7	magenta
110	5	cyan
111	1	black

Table 6-2. Pixel Value to Color Index Mapping for 16-color Screens

Pixel Value	Color Index	Color
0000	0	white
0001	2	red
0010	3	green
0011	6	yellow
0100	4	blue
0101	7	magenta
0110	5	cyan
0111	8	low white
1000	9	grey
1001	10	light red
1010	11	light green
1011	14	light yellow
1100	12	light blue
1101	15	light magenta
1110	13	light cyan
1111	1	black

Note: A pixel value of 0 maps to the background color.

In addition to the MFDB, Copy Raster also takes a rectangle as an argument. This allows operations on a specified portion of the raster area. A rectangle is specified by the x,y coordinates of its upper left and lower right vertices.

COORDINATE SYSTEMS

A sample single-plane memory form with a form width of 16 pixels, a form height of 8 pixels, and a highlighted rectangle with corners of (3,1) and (6,5) is shown in Figure 6-3.

306B

Figure 6-2. Standard Forms

Figure 6-3. Sample Single Plane Memory Form

LOGIC OPERATIONS

To provide greatest flexibility, raster operations subject to a logic operation take the operation as an argument rather than using the logic operation associated with vector primitives. In addition, the operations available are greatly expanded to allow more flexibility. Table 6-3 lists the available operations with the following conventions:

- S = pixel value (0 or 1) of source pixel
- D = pixel value (0 or 1) of destination pixel
- D'= destination pixel value after the logical operation

Table 6-3. Raster Operation Logic Operations

Mode	Definition	
0	D'= 0	
1	D'= S AND D	
2	D'= S AND [NOT D]	
3	D'= S	(Replace mode)
4	D'= [NOT S] AND D	(Erase mode)
5	D'= D	
6	D'= S XOR D	(Xor mode)
7	D'= S OR D	
8	D'= NOT [S OR D]	
9	D'= NOT [S XOR D]	
10	D'= NOT D	
11	D'= S OR [NOT D]	
12	D'= NOT S	
13	D'= [NOT S] OR D	
14	D'= NOT [S AND D]	
15	D'= 1	

COPY RASTER, OPAQUE

This function copies a rectangular raster area from source form to destination form using the logic operation the application specifies. If the source and destination forms are the same, and the rectangles overlap, GEM VDI copies so that the source rectangle is not changed until GEM VDI processes the corresponding area in the destination. No rotation or transformation occurs as a result of this function; the copy is pixel for pixel.

If the source and destination rectangles are not the same size, GEM VDI uses the destination as a pointer and uses the source for the size. The Extended Inquire function returns scaling ability. The source and destination forms must be in device-specific form; see "Transform Form" later in this section.

Input

contrl(0) --	Opcode = 109.
contrl(1) --	Number of input vertices = 4.
contrl(3) --	Length of intin array = 1.
contrl(6) --	Device handle.
contrl(7-8)--	Double-word address of the source Memory Form Definition Block.
contrl(9-10)-	Double-word address of the destination Memory Form Definition Block.
intin(0) --	Logic operation (refer to "Introduction" in this section).
ptsin(0) --	x-coordinate of corner of source rectangle in RC/NDC.
ptsin(1) --	y-coordinate of corner of source rectangle in RC/NDC.
ptsin(2) --	x-coordinate of corner diagonally opposite corner selected in ptsin(0) of source rectangle in RC/NDC.
ptsin(3) --	y-coordinate of corner diagonally opposite corner selected in ptsin(1) of source rectangle in RC/NDC.

	ptsin(4)	--	x-coordinate of corner of destination rectangle in RC/NDC.
	ptsin(5)	--	y-coordinate of corner of destination rectangle in RC/NDC.
	ptsin(6)	--	x-coordinate of corner of destination rectangle in RC/NDC.
	ptsin(7)	--	y-coordinate of corner of destination rectangle in RC/NDC.

Output	contrl(2)	--	Number of output vertices = 0.
	contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
vro_cpyfm( handle, wr_mode, pxyarray,
    psrcMFDB, pdesMFDB )
```

Data Types

```
WORD vro_cpyfm ( );
WORD handle;
WORD wr_mode;
WORD pxyarray[8];
WORD *psrcMFDB;
WORD *pdesMFDB;
```

Input Arguments

```
handle = contrl[6]
wr_mode = intin[0]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
  .
  .
  .
pxyarray[7] = ptsin[7]
psrcMFDB = contrl[7-8]
pdesMFDB = contrl[9-10]
```

COPY RASTER, TRANSPARENT

This function copies a monochrome rectangular raster area from source form to a color area. A writing mode and color indices for both 0's and 1's are specified in the intin array.

If the source and destination rectangles are not the same size, GEM VDI uses the source rectangle for the size and the upper left corner of the destination rectangle for the initial destination location.

Transfer of information from the source to the destination is controlled by the specified writing mode as described below. See Table 5-1 for a binding of the available writing modes.

Replace Mode

Replace mode will result in a replacement of all pixels in the destination rectangle. The foreground color index specified in intin(1) will be output to all pixels associated with source locations which are set to a one. The background color index specified in intin(2) will be output to all pixels associated with source locations which are set to a zero.

Transparent Mode

Transparent mode only affects the pixels associated with a source value of one. Those pixels are set to the foreground color whose index is specified in intin(1). The color index specified in intin(2) is not used.

XOR Mode

In XOR mode, the monochrome raster source area is logically XORed with each plane of the destination. The color indices specified in intin(1) and intin(2) are not used.

Reverse Transparent Mode

Reverse Transparent mode only affects the pixels associated with a source value of zero. Those pixels are set to the background color whose index is specified in intin(2). The color index specified in intin(1) is not used.

Input

contrl(0) --	Opcode = 121.
contrl(1) --	Number of input vertices = 4.
contrl(3) --	Length of intin array = 3.
contrl(6) --	Device handle.
contrl(7-8)--	Double-word address of the source Memory Form Definition Block.
contrl(9-10)-	Double-word address of the destination Memory Form Definition Block.
intin(0) --	Writing Mode.
intin(1) --	Color index for 1s in data.
intin(2) --	Color index for 0s in data.
ptsin(0) --	x-coordinate of corner of source rectangle in RC/NDC.
ptsin(1) --	y-coordinate of corner of source rectangle in RC/NDC.
ptsin(2) --	x-coordinate of corner diagonally opposite corner selected in ptsin(0) of source rectangle in RC/NDC.
ptsin(3) --	y-coordinate of corner diagonally opposite corner selected in ptsin(1) of source rectangle in RC/NDC.
ptsin(4) --	x-coordinate of corner of destination rectangle in RC/NDC.
ptsin(5) --	y-coordinate of corner of destination rectangle in RC/NDC.
ptsin(6) --	x-coordinate of corner of destination rectangle in RC/NDC.
ptsin(7) --	y-coordinate of corner of destination rectangle in RC/NDC.

312

Output contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name

```
vrt_cpyfm( handle, wr_mode, pxyarray,
    psrcMFDB, pdesMFDB, color_index )
```

Data Types

```
WORD vrt_cpyfm ( );
WORD handle;
WORD wr_mode;
WORD pxyarray[8];
WORD *psrcMFDB;
WORD *pdesMFDB;
WORD color_index[2];
```

Input Arguments

```
handle = contrl[6]
wr_mode = intin[0]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
  .
  .
  .
pxyarray[7] = ptsin[7]
psrcMFDB = contrl[7-8]
pdesMFDB = contrl[9-10]
color_index[0] = intin[1]
color_index[1] = intin[2]
```

TRANSFORM FORM

This function transforms a raster area from standard format to device-specific format or from device-specific to standard format. The operation is a toggle, changing the current state.

The number of planes specified in the source MFDB determines the number transformed. The source format flag is toggled and placed in the destination. The user is required to ensure that the other parameters in the destination MFDB are correct.

Input

contrl(0) --	Opcode = 110.
contrl(1) --	Number of input vertices = 0.
contrl(3) --	Length of intin array = 0.
contrl(6) --	Device handle.
contrl(7-8) --	Double-word address of the source MFDB.
contrl(9-10) -	Double-word address of the destination MFDB.

Output

contrl(2) --	Number of output vertices = 0.
contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name

```
vr_trnfm( handle, psrcMFDB, pdesMFDB )
```

Data Types

```
WORD vr_trnfm ( );
WORD handle;
WORD *psrcMFDB;
WORD *pdesMFDB;
```

Input Arguments

```
handle = contrl[6]
psrcMFDB = contrl[7-8]
pdesMFDB = contrl[9-10]
```

GET PIXEL	This function returns a pixel value and a color index for the pixel specified by ptsin(0), ptsin(1). **Note:** Color index 0 is the background color. It may or may not map to pixel value 0 in device-specific form. Refer to Tables 6-1 and 6-2 for the colors and values. Standard form always maps color index 0 to pixel value 0.
Input	contrl(0) -- Opcode = 105. contrl(1) -- Number of input vertices = 1. contrl(3) -- Length of intin array = 0. contrl(6) -- Device handle. ptsin(0) -- x-coordinate of pixel in RC/NDC units. ptsin(1) -- y-coordinate of pixel in RC/NDC units.
Output	contrl(2) -- Number of output vertices = 0. contrl(4) -- Length of intout array = 2. intout(0) -- Pixel value. intout(1) -- Color index.

315

C BINDING

Procedure Name v_get_pixel(handle, x, y, pel, index)

Data Types

```
WORD v_get_pixel( );
WORD handle;
WORD x;
WORD y;
WORD *pel;
WORD *index;
```

Input Arguments

handle = contrl[6]
x = ptsin[0]
y = ptsin[1]

Output Arguments

pel = intout[0]
index = intout[1]

End of Section 6

3/6

Section 7
INPUT FUNCTIONS

INTRODUCTION

The input functions allow user interactions with the application program. Many of the input functions support two modes: request and sample. In request mode, the driver waits until an input event occurs before returning. In sample mode, the driver returns the current status or location of the input device without waiting.

SET INPUT MODE

This function sets the input mode for the following specified logical input devices to request or sample:

- o locator
- o valuator
- o choice
- o string

Select the input mode in intin(1).

Input

contrl(0)	--	Opcode = 33.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 2.
contrl(6)	--	Device handle.
intin(0)	--	Logical input device. 1 = locator 2 = valuator 3 = choice 4 = string
intin(1)	--	Input mode. 1 = request 2 = sample

317

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 1.
	intout(0) --	Input mode selected.

C BINDING

Procedure Name vsin_mode(handle, dev_type, mode)

Data Types

```
WORD vsin_mode ( );
WORD handle;
WORD dev_type;
WORD mode;
```

Input Arguments

```
handle = contrl[6]
dev_type = intin[0]
mode = intin[1]
```


318

INPUT LOCATOR, REQUEST MODE

This function returns the position of the specified locator device. Upon entry to the locator routine, the current cursor form is displayed at the initial coordinate. The graphic cursor is tracked with the input device until a terminating event occurs, which can result from the user pressing a key or a button on a mouse. GEM VDI removes the cursor when the terminating event occurs. Typically, the arrow keys move the cursor in large jumps when used without the Shift key and in pixel increments when used with the Shift key.

This function always displays a cursor on the screen, even if the cursor is currently obscured or hidden.

Note: If both a keyboard and another locator device are available, the cursor is tracked by input from either, giving the user maximum flexibility.

Input

contrl(0)	--	Opcode = 28.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
ptsin(0)	--	Initial x-coordinate of locator in NDC/RC units.
ptsin(1)	--	Initial y-coordinate of locator in NDC/RC units.

Output	contrl(2) --	Number of output vertices = 1.
	contrl(4) --	Length of intout array = 1.
	intout(0) --	Locator terminator. The low byte contains a character terminator. For keyboard-terminated locator input, this is the ASCII character code of the key struck to terminate input. For nonkeyboard-terminated input (tablet, mouse, and so on), valid locator terminators begin with 20 Hex (space) and increase from there. For instance, if the puck on a tablet has 4 buttons, the first button must generate a 20 Hex as a terminator, the second a 21 Hex, the third a 22 Hex, and the fourth a 23 Hex.
	ptsout(0) --	Final x-coordinate of locator in NDC/RC units.
	ptsout(1) --	Final y-coordinate of locator in NDC/RC units.

C BINDING

Procedure Name	vrq_locator(handle, x, y, &xout, &yout, &term)
Data Types	WORD vrq_locator ( ); WORD handle; WORD x, y; WORD xout; WORD yout; WORD term;
Input Arguments	handle = contrl[6] x = ptsin[0] y = ptsin[1]
Output Functions	xout = ptsout[0] yout = ptsout[1] term = intout[0]

INPUT LOCATOR, SAMPLE MODE

This function returns the position in NDCs of the specified locator device. Upon entry to the locator routine, no cursor is displayed. (Use Show Cursor to display the cursor.) Input is sampled. If the cursor position has changed, GEM VDI returns the cursor position and contrl(2) is set to 1. Contrl(4) is set to 0. If a terminating event occurred, GEM VDI returns a character and contrl(4) is set to 1. Contrl(2) is set to 0.

Note: If both a keyboard and another locator device are available, the input comes from either, giving the user maximum flexibility.

Input

contrl(0)	--	Opcode = 28.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
ptsin(0)	--	Initial x-coordinate of locator in NDC/RC units.
ptsin(1)	--	Initial y-coordinate of locator in NDC/RC units.

Output

contrl(2) -- Number of output vertices.

1 = coordinate changed
0 = no coordinate changed

contrl(4) -- Length of intout array.

0 = no keypress character
1 = keypress character returned

322

Table 7-1. Sample Mode Status Returned

Event	Control Array (2)	(4)
Coordinates change.	1	0
Key pressed; coordinates not changed from what was pressed.	0	1
No input.	0	0
Key pressed; coordinates changed.	1	1

intout(0) -- Locator keypress if keypress occurs.

This information is the same as for Input Locator, Request Mode function.

ptsout(0) -- New x-coordinate of locator in NDC/RC units.

ptsout(1) -- New y-coordinate of locator in NDC/RC units.

C BINDING

Procedure Name	status = vsm_locator(handle, x, y, &xout, &yout, &term)
Data Types	WORD status; WORD vsm_locator ( ); WORD handle; WORD x, y; WORD xout; WORD yout; WORD term;
Input Arguments	handle = contrl[6] x = ptsin[0] y = ptsin[1]
Output Arguments	status = contrl[2] | (contrl[4] << 1) xout = ptsout[0] yout = ptsout[1] term = intout[0]

324

INPUT VALUATOR, REQUEST MODE

This function returns the value of the valuator device. The initial value of the valuator is incremented or decremented until a terminating character is struck. Valuator keys are typically the up-arrow and down-arrow keys. Valuator numbers range from 1 to 100. Typical implementation of the up-arrow and down-arrow keys is as follows:

- Pressing the up-arrow key adds ten to the valuator.
- Pressing the down-arrow key subtracts ten from the valuator.
- Pressing the up-arrow key with the Shift key adds one to the valuator.
- Pressing the down-arrow key with the Shift key subtracts one from the valuator.

Note: This function is not required and may not be available on all devices.

Input

```
contrl(0) --   Opcode = 29.
contrl(1) --   Number of input vertices = 0.
contrl(3) --   Length of intin array = 1.
contrl(6) --   Device handle.

intin(0)  --   Initial value.
```

Output

```
contrl(2) --   Number of output vertices = 0.
contrl(4) --   Length of intout array = 2.

intout(0) --   Output value.
intout(1) --   Terminator.
```

C BINDING

Procedure Name

```
vrq_valuator( handle, valuator_in,
    &valuator_out, &terminator )
```

Data Types

```
WORD vrq_valuator ( );
WORD handle;
WORD valuator_in;
WORD valuator_out;
WORD terminator;
```

Input Arguments

```
handle = contrl[6]
valuator_in = intin[0]
```

Output Arguments

```
valuator_out = intout[0]
terminator = intout[1]
```

INPUT VALUATOR, SAMPLE MODE

This function returns the current value of the valuator device. The valuator device is sampled. If the valuator has changed, GEM VDI increments or decrements the valuator value as required. If a terminating event occurs, GEM VDI returns the value. If nothing happens, GEM VDI returns no value. Valuator numbers range from 1 to 100. The suggested keys are the same as for Input Valuator, Request Mode.

Note: This function is not required and may not be available on all devices.

Input

contrl(0) -- Opcode = 29.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Initial value.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array.

0 = nothing happened
1 = valuator changed
2 = keypress character

intout(0) -- New valuator value.
intout(1) -- Keypress, if keypress event occurred.

C BINDING

Procedure Name	vsm_valuator(handle, val_in, &val_out, &term, &status)
Data Types	WORD vsm_valuator ( ); WORD handle; WORD val_in; WORD val_out; WORD term; WORD status;
Input Arguments	handle = contrl[6] val_in = intin[0]
Output Arguments	val_out = intout[0] term = intout[1] status = contrl[4]

INPUT CHOICE, REQUEST MODE

This function returns the choice status of the selected choice device. Input is sampled until a key is pressed. If it is a valid choice key, GEM VDI returns its value. Otherwise, GEM VDI returns the initial choice number. Choice numbers range from 1 to a device-dependent maximum value.

Note: This function is not required and may not be available on all devices.

Input

contrl(0)	--	Opcode = 30.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Initial choice number.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Choice number.

C BINDING

Procedure Name

```
vrq_choice( handle, ch_in, &ch_out )
```

Data Types

```
WORD vrq_choice ( );
WORD handle;
WORD ch_in;
WORD *ch_out;
```

Input Arguments

```
handle = contrl[6]
ch_in = intin[0]
```

Output Arguments

```
*ch_out = intout[0]
```

INPUT CHOICE, SAMPLE MODE	This function returns the choice status of the selected choice device. Upon entry to the routine, GEM VDI samples input. If input is available and is a valid choice key, GEM VDI returns it. Choice numbers range from 1 to a device-dependent maximum value. **Note:** This function is not required and may not be available on all devices.

Input

```
contrl(0) --   Opcode = 30.
contrl(1) --   Number of input vertices = 0.
contrl(3) --   Length of intin array = 0.
contrl(6) --   Device handle.
```

Output

```
contrl(2) --   Number of output vertices = 0.
contrl(4) --   Choice status.

               0 = nothing happened
               1 = sample successful

intout(0) --   Choice number if sample suc-
               cessful, 0 if unsuccessful.
```

C BINDING

Procedure Name

```
status = vsm_choice( handle, &choice )
```

Data Types

```
WORD status;
WORD vsm_choice ( );
WORD handle;
WORD choice;
```

Input Arguments

```
handle = contrl[6]
```

Output Arguments

```
choice = intout[0]
status = contrl[4]
```

INPUT STRING, REQUEST MODE

This function returns a string from the specified device. Input is accumulated until GEM VDI encounters a carriage return or the intout array is full. If the application enables echo mode, text will be echoed to the screen with the current text attributes using the vertex passed in the ptsin array as the justification point.

If the number in intin(0) is negative, the values in intout will conform to the standard keyboard defined in Appendix D. In this case, the absolute value of intin(0) is used as the maximum intout size.

Note: Echoing of input is not required and may not be available on all devices.

Input

contrl(0)	--	Opcode = 31.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 2.
contrl(6)	--	Device handle.
intin(0)	--	Maximum string length.
intin(1)	--	Echo mode. 0 = no echo 1 = echo input characters at position specified
ptsin(0)	--	x-coordinate of echo area in NDC/RC units.
ptsin(1)	--	y-coordinate of echo area in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array.
intout	--	Output string returned in ADE.

C BINDING

Procedure Name

```
vrq_string( handle, max_length, echo_mode,
    echo_xy, &string )
```

Data Types

```
WORD vrq_string ( );
WORD handle;
WORD max_length;
WORD echo_mode;
WORD echo_xy[2];
BYTE string[max_length+1];
```

Input Arguments

```
handle = contrl[6]
max_length = intin[0]
echo_mode = intin[1]
echo_x,y = ptsin[0-1]
```

Output Arguments

string = intout

Note: The BYTE array elements contain the eight least significant bits of the intout array elements. The array is terminated with a null byte. The length of the output variable string includes an additional byte for the terminating null.

INPUT STRING, SAMPLE MODE

This function returns a string from the specified device. Upon entry to the routine, GEM VDI samples input. If data is available, it is accumulated, and GEM VDI samples the input again. Input is accumulated until one of the following events occurs:

- Data is no longer available.
- A carriage return is encountered.
- The intout buffer is full.

Note: If the string will always be terminated with RETURN, use Input String, Request Mode.

If the number in intin(0) is negative, the values in intout will conform to the standard keyboard defined in Appendix D. In this case, the absolute value of intin(0) is used as the maximum intout size.

Input

contrl(0)	--	Opcode = 31.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 2.
contrl(6)	--	Device handle.
intin(0)	--	Maximum string length.
intin(1)	--	Echo mode. 0 = no echo 1 = echo input characters
ptsin(0)	--	x-coordinate of echo area in NDC/RC units.
ptsin(1)	--	y-coordinate of echo area in NDC/RC units.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of output string. 0 = sample unsuccessful (characters not available) >0 = sample successful (characters available)
intout	--	Output string, if sample successful.

C BINDING

Procedure Name

```
status = vsm_string( handle, max_length,
    echo_mode, echo_xy, &string )
```

Data Types

```
WORD vsm_string ( );
WORD handle;
WORD max_length;
WORD echo_mode;
WORD echo_xy[2];
BYTE string[max_length+1];
WORD status;
```

Input Arguments

handle = device handle
max_length = intin[0]
echo_mode = intin[1]
echo_xy = ptsin[0-1]

Output Arguments

string = intout
status = contrl[4]

Note: The BYTE array elements contain the eight least significant bits of the intout array elements. The array is terminated with a null byte. The length of the output variable string includes an additional byte for the terminating null.

SET MOUSE FORM

This function redefines the cursor pattern displayed during locator input or at any time the cursor is shown (see the discussion of the Show Cursor function later in this section).

For the cursor mask and data, bit 15 of word 1 is the upper left bit of the pattern. Bit 0 of word 16 is the lower right bit of the pattern. Bit zero is the Least Significant Bit of the word.

The hot spot is the location of the pixel (relative to the upper left pixel of the mouse form) that lies over the pixel whose address is returned by the input locator function.

The mouse form is drawn as follows:

1. The data under the mouse form is saved so that it can be restored when the cursor moves.

2. 1s in the mask cause the corresponding pixel to be set to the color index defined in intin(3).

3. 1s in the mouse form data cause the corresponding pixel to be set to the color index defined in intin(4).

Input

```
Contrl(0) --   Opcode = 111.
Contrl(1) --   Number of input vertices = 0.
Contrl(3) --   Length of intin array = 37.
Contrl(6) --   Device Handle.

intin(0)   --  x-coordinate of hot spot.
intin(1)   --  y-coordinate of hot spot.
intin(2)   --  Reserved for future use, must
               be 1.
intin(3)   --  Mask color index, normally 0.
intin(4)   --  Data color index, normally 1.
intin(5-20)-   16 words of 16 bit cursor mask.
intin(21-36)-  16 words of 16 bit cursor data.
```

Output

Contrl(2) -- Number of output vertices = 0.
Contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name

```
vsc_form( handle, pcur_form )
```

Data Types

```
WORD vsc_form ( );
WORD handle;
WORD pcur_form[37];
```

Input Arguments

```
handle = contrl[6]
pcur_form[0] = intin[0]
  .
  .
  .
pcur_form[36] = intin[36]
```

EXCHANGE TIMER INTERRUPT VECTOR

With this function, the application can perform some action each time a timer tick occurs.

The input to this function is a two-word pointer in contrl(7) and contrl(8). The pointer indicates the starting address of the code to receive control when a timer tick occurs. The address of the old timer routine is returned in contrl(9) and contrl(10).

The application-dependent code is invoked with a processor-dependent call instruction. When this is complete, the application should perform a processor-dependent return instruction.

It is the responsibility of the application-dependent code to save and restore any registers used.

When the application code is invoked, interrupts are disabled. The application should not enable interrupts.

See Appendix E for processor specific instructions and register names.

The number of milliseconds per timer tick is returned in intout(0).

Input

contrl(0)	--	Opcode = 118.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
contrl(7-8)	-	Address of application timer routine.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 1.
contrl(9-10)	-	Address of the old timer routine.
intout(0)	--	Milliseconds per tick.

C BINDING

Procedure Name

```
vex_timv( handle, tim_addr, otim_addr,
    &tim_conv )
```

Data Types

```
WORD vex_timv( );
WORD handle;
WORD *tim_addr;
WORD *otim_addr;
WORD tim_conv;
```

Input Arguments

```
handle = contrl[6]
tim_addr = contrl[7-8]
```

Output Arguments

```
otim_addr = contrl[9-10]
tim_conv = intout[0]
```

SHOW CURSOR

This function displays the current cursor. The cursor moves on the display surface based on information input from a mouse.

The Show Cursor function and the Hide Cursor functions are closely related. Once the cursor is visible, a single Hide Cursor causes the cursor to disappear. GEM VDI keeps track of the number of times the Hide Cursor function is called. The Show Cursor function must be called the same number of times for the cursor to reappear. For example, if the Hide Cursor function is called four times, the Show Cursor function must be called four times for the cursor to appear.

The Show Cursor function does, however, provide a reset flag in intin(0). If intin(0) is zero, the cursor appears on the screen, regardless of the number of Hide Cursor calls. A nonzero value for intin(0) affects the Show Cursor function as described in the preceding paragraph.

Input

contrl(0)	--	Opcode = 122.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Reset flag.
		0 = ignore number of Hide Cursor calls
		nonzero = normal Show Cursor functionality

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name	v_show_c(handle, reset)
Data Types	WORD v_show_c ( ); WORD handle; WORD reset
Input Arguments	handle = contrl[6] reset = intin[0]

HIDE CURSOR	This function removes the cursor from the display surface. This state is the default condition set at Open Workstation. The cursor can appear in a new position when the application calls the Show Cursor function because GEM VDI updates the position based on information input from a mouse. Refer to the Show Cursor function for a description of how the number of Hide Cursor calls affects the Show Cursor function.
Input	contrl(0) -- Opcode = 123. contrl(1) -- Number of input vertices = 0. contrl(3) -- Length of intin array = 0. contrl(6) -- Device handle.
Output	contrl(2) -- Number of output vertices = 0. contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name	v_hide_c(handle)
Data Types	WORD v_hide_c ( ); WORD handle;
Input Arguments	handle = contrl[6]

341

SAMPLE MOUSE BUTTON STATE

This function returns the current state of the mouse buttons. The leftmost mouse button is returned in the Least Significant Bit of the word. A bit value of 1 indicates the key is currently depressed; a bit value of 0 indicates the key is up.

This function also returns the current (x,y) position of the cursor.

Input

contrl(0)	--	Opcode = 124.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.

Output

contrl(2)	--	Number of output vertices = 1.
contrl(4)	--	Length of intout array = 1.
intout(0)	--	Mouse button state.
ptsout(0)	--	x position of cursor in NDC/RC units.
ptsout(1)	--	y position of cursor in NDC/RC units.

C BINDING

Procedure Name

```
vq_mouse( handle, &pstatus, &x, &y )
```

Data Types

```
WORD vq_mouse ( );
WORD handle;
WORD pstatus;
WORD x, y;
```

Input Arguments

```
handle = contrl[6]
```

Output Arguments

```
pstatus = intout[0]
x = ptsout[0]
y = ptsout[1]
```

EXCHANGE BUTTON CHANGE VECTOR

This function allows the application to perform some action each time the state of the mouse buttons changes. The application receives control after the button state is decoded, but before the driver button state changes.

The input to this function is a two-word pointer in contrl(7) and contrl(8), which indicates the starting address of the code to receive control when the mouse button state changes. Contrl(9) and contrl(10) return a two-word pointer to the old mouse routine.

Control is passed to the specified address whenever the mouse button state changes. The application code is invoked via a processor-dependent call instruction with a processor-dependent register containing the mouse button keys. Keys are encoded by the same rules that apply to the Sample Mouse Button State function. When complete, the application-dependent code should do a processor-dependent return instruction with the mouse button state the driver is to store in the same register. This gives the application the opportunity to alter the buttons before they are used by the driver.

It is the responsibility of the application-dependent code to save and restore any registers used.

When the application code is invoked, interrupts are disabled. The application should not enable interrupts.

See Appendix E for processor-specific instructions and register names.

Input

Contrl(0) --	Opcode = 125.
Contrl(1) --	Number of input vertices = 0.
Contrl(3) --	Length of intin array = 0.
Contrl(6) --	Device handle.
Contrl(7-8)-	Address of application mouse button state change routine.

Output	Contrl(2) --	Number of output vertices = 0.
	Contrl(4) --	Length of intout array = 0.
	Contrl(9-10) -	Address of old mouse button state change routine.

C BINDING

Procedure Name vex_butv(handle, pusrcode, psavcode)

Data Types

```
WORD vex_butv ( );
WORD handle;
WORD *pusrcode;
WORD *psavcode;
```

Input Arguments

handle = contrl[6]
pusrcode = contrl[7-8]

Output Arguments

psavcode = contrl[9-10]

344

EXCHANGE MOUSE MOVEMENT VECTOR

This function allows the application to perform some action each time the mouse moves to a new location. The application receives control after the x,y address is computed, but before the current mouse position in the driver is updated or the mouse form is actually redrawn on the screen.

The input to this function is a two-word pointer in contrl(7) and contrl(8), which indicates the starting address of the code to receive control when the mouse moves. A two-word pointer to the address of the old mouse movement routine is returned in contrl(9) and contrl(10).

When the mouse moves, the application-dependent code is invoked via a processor-dependent call instruction. The new x and y locations are contained in a pair of processor-dependent registers. Upon completion, the application-dependent code should do a processor-dependent return instruction with the x,y mouse position the driver is to store in the appropriate hardware registers. This procedure gives the opportunity to alter the x,y position before it is used by the driver.

It is the responsibility of the application-dependent code to save and restore any registers used.

When the application code is invoked, interrupts are disabled. The application should not enable interrupts.

See Appendix E for processor-specific instructions and register names.

Input	Contrl(0) --	Opcode = 126.
	Contrl(1) --	Number of input vertices = 0.
	Contrl(3) --	Length of intin array = 0.
	Contrl(6) --	Device handle.
	Contrl(7-8)-	Address of application mouse movement routine.

Output	Contrl(2) --	Number of output vertices = 0.
	Contrl(4) --	Length of intout array = 0.
	Contrl(9-10) -	Address of the old mouse movement routine.

C BINDING

Procedure Name vex_motv(handle, pusrcode, psavcode)

Data Types

```
WORD vex_motv ( );
WORD handle;
WORD *pusrcode;
WORD *psavcode;
```

Input Arguments

handle = contrl[6]
pusrcode = contrl[7-8]

Output Arguments

psavcode = contrl[9-10]

EXCHANGE CURSOR CHANGE VECTOR

This function allows the application to perform some action each time the cursor is drawn. The application can completely take over drawing the cursor or can perform some action and have GEM VDI draw the cursor. Control is passed to the application whenever the cursor position should be updated.

The input to this function is a two-word pointer in contrl(7) and contrl(8), which indicates the starting address of the code to receive control when a cursor is drawn. The address of the old cursor draw routine is returned in contrl(9) and contrl(10).

The application-dependent code is invoked with a processor-dependent call instruction. The x,y position at which the cursor should be drawn is contained in a pair of processor-dependent registers. If the application-dependent code does not draw its own cursor, a processor-dependent call should be performed to the address returned in contrl(9) and contrl(10). This will cause GEM VDI to draw a cursor. When it is done, the application should perform a processor-dependent return instruction.

It is the responsibility of the application-dependent code to save and restore any registers used. The GEM VDI cursor draw routine preserves the contents of all registers.

When the application code is invoked, interrupts are disabled. The application should not enable interrupts.

See Appendix E for processor-specific instructions and register names.

Input	contrl(0) --	Opcode = 127.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(6) --	Device handle.
	contrl(7-8)-	Address of application cursor draw routine.
Output	contrl(2) --	Number of output vertices = 0
	contrl(4) --	Length of intout array = 0
	contrl(9-10) -	Address of the old cursor draw routine.

C BINDING

Procedure Name vex_curv(handle, pusrcode, psavcode)

Data Types

```
WORD vex_curv ( );
WORD handle;
WORD *pusrcode;
WORD *psavcode;
```

Input Arguments

handle = contrl[6]
pusrcode = contrl[7-8]

Output Arguments

psavcode = contrl[9-10]

348

SAMPLE KEYBOARD STATE INFORMATION

This function returns the current state of the keyboard's Control, Shift, and Alt keys. These values are returned as a bit-encoded value in intout(0). The keys are assigned to bits as follows:

Bit 0 - right Shift Key
Bit 1 - left Shift Key
Bit 2 - Control Key
Bit 3 - Alt Key

Bit 0 is the Least Significant Bit of the word. A bit value of zero indicates the key is up, a bit value of 1 indicates the key is depressed.

Input

contrl(0) -- Opcode = 128.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Keyboard state.

C BINDING

Procedure Name vq_key_s(handle, &pstatus)

Data Types

```
WORD vq_key_s ( );
WORD handle;
WORD pstatus;
```

Input Arguments handle = contrl[6]

Output Arguments pstatus = intout[0]

End of Section 7

Section 8
INQUIRE FUNCTIONS

INTRODUCTION

Inquire functions return the current settings for device-specific attributes.

EXTENDED INQUIRE

This function returns additional device-specific information not included in the Open Workstation call. The value of intin(0) determines if GEM VDI returns the values returned at Open Workstation or an extended set of device-specific information. Refer to Section 3, "Control Functions," for more information about intout values for the Open Workstation function.

Note that 6 vertices and 45 intouts are always returned, although some values are undefined for the extended device information.

Input

contrl(0)	--	Opcode = 102.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Information type.
		0 = Open Workstation values
		1 = Extended Inquire values

Output

	contrl(2) --	Number of output vertices = 6.
	contrl(4) --	Length of intout array = 45.
	intout(0) --	Type of screen. 0 -- not screen 1 -- separate alpha and graphic controllers and separate video screens 2 -- separate alpha and graphic controllers with a common video screen 3 -- common alpha and graphic controller with separate image memory 4 -- common alpha and graphic controller with common image memory
	intout(1) --	Number of background colors available in color palette. On some devices this may be different from the number of colors returned from Open Workstation, intout(39).
	intout(2) --	Text effects supported. (See "Set Graphic Text Special Effects" in Section 5 for values.)
	intout(3) --	Scale rasters. 0 = scaling not possible 1 = scaling possible
	intout(4) --	Number of planes.
	intout(5) --	Lookup table supported. 0 = table supported 1 = table not supported
	intout(6) --	Performance factor, number of 16 x 16 pixel raster ops per second.
	intout(7) --	Contour fill capability.
	intout(8) --	Character rotation ability. 0 = none 1 = 90-degree increments only 2 = arbitrary angles

intout(9) --	Number of writing modes available.
intout(10)--	Highest level of input mode available. 0 = none 1 = request 2 = sample
intout(11)--	Text alignment capability flag. 0 = no 1 = yes
intout(12)--	Inking capability flag. 0 = device cannot ink 1 = device can ink
intout(13)--	Rubberbanding capability flag. 0 = no 1 = capable of rubberband lines 2 = capable of both rubberband lines and rectangles
intout(14)--	Maximum vertices for Polyline, Polymarker, or Filled Area. -1 = no maximum
intout(15)--	Maximum intin. -1 = no maximum
intout(16)--	Number of keys available on the mouse.
intout(17)--	Styles for wide lines. 0 = no 1 = yes
intout(18)--	Writing modes for wide lines.
intout(19-44)-	Reserved, contains zeros.
ptsout(0-11) -	Reserved, contains zeros.

C BINDING

Procedure Name vq_extnd(handle, owflag, work_out)

Data Types

```
WORD vq_extnd ( );
WORD handle;
WORD owflag;
WORD work_out[57]
```

Input Arguments

```
handle = contrl[6]
owflag = intin[0]
```

Output Arguments

```
work_out[0] = intout[0]
  .
  .
work_out[44] = intout[44]
work_out[45] = ptsout[0]
  .
  .
work_out[56] = ptsout[11]
```

INQUIRE COLOR REPRESENTATION

This function returns either the requested or the actual value of the specified color index in RGB units. Both the set and realized values are available. If the selected index is out of range, GEM VDI returns -1 in intout(0).

Input

contrl(0)	--	Opcode = 26.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 2.
contrl(6)	--	Device handle.
intin(0)	--	Requested color index.
intin(1)	--	Set or realized flag. 0 = set (return color values requested) 1 = realized (return color values realized on device)

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 4.
intout(0)	--	Color index.
intout(1)	--	Red intensity (in tenths of percent 0-1000).
intout(2)	--	Green intensity.
intout(3)	--	Blue intensity.

C BINDING

Procedure Name	vq_color(handle, color_index, set_flag, rgb)
Data Types	WORD vq_color ( ); WORD handle; WORD color_index; WORD set_flag; WORD rgb[3];
Input Arguments	handle = contrl[6] color_index = intin[0] set_flag = intin[1]
Output Arguments	rgb[0] = intout[1] rgb[1] = intout[2] rgb[2] = intout[3]

INQUIRE CURRENT POLYLINE ATTRIBUTES	This function reports the current setting of all attributes that affect polylines, such as line type, line color, line width, end styles, and writing mode.

Input

contrl(0) --	Opcode = 35.
contrl(1) --	Number of input vertices = 0.
contrl(3) --	Length of intin array = 0.
contrl(6) --	Device handle.

Output

contrl(2) --	Number of output vertices = 1.
contrl(4) --	Length of intout array = 5.
intout(0) --	Current polyline line type. (Refer to Set Polyline Line Type function.)
intout(1) --	Current polyline line color index.
intout(2) --	Current writing mode. (Refer to the Set Writing Mode function.)
intout(3) --	End style for beginning point of polyline.
intout(4) --	End style for ending point of polyline.
ptsout(0) --	Current line width, in current coordinate system.
ptsout(1) --	0.

356

C BINDING

Procedure Name	vql_attributes(handle, attrib)
Data Types	WORD vql_attributes ( ); WORD handle; WORD attrib[4];
Input Arguments	handle = contrl[6]
Output Arguments	attrib[0] = intout[0] attrib[1] = intout[1] attrib[2] = intout[2] attrib[3] = ptsout[0]

INQUIRE CURRENT POLYMARKER ATTRIBUTES	This function reports the current setting of all attributes that affect polymarkers, such as marker type, marker color, marker height, and writing mode.

Input	contrl(0) --	Opcode = 36.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 1.
	contrl(4) --	Length of intout array = 3.
	intout(0) --	Current polymarker marker type. (Refer to Set Polymarker Type function.)
	intout(1) --	Current polymarker marker color index.
	intout(2) --	Current writing mode. (Refer to the Set Writing Mode function for description.)
	ptsout(0) --	Current polymarker width, in current coordinate system.
	ptsout(1) --	Current polymarker height, in current coordinate system.

C BINDING

Procedure Name	vqm_attributes(handle, attrib)
Data Types	WORD vqm_attributes ( ); WORD handle; WORD attrib[4];
Input Arguments	handle = contrl[6]
Output Arguments	attrib[0] = intout[0] attrib[1] = intout[1] attrib[2] = intout[2] attrib[3] = ptsout[1]

INQUIRE CURRENT FILL AREA ATTRIBUTES	This function reports the current setting of all attributes that affect fill areas, such as interior style, fill color, fill style index, and writing mode.
Input	contrl(0) -- Opcode = 37. contrl(1) -- Number of input vertices = 0. contrl(3) -- Length of intin array = 0. contrl(6) -- Device handle.
Output	contrl(4) -- Number of output vertices = 0. contrl(6) -- Length of intout array = 5. intout(0) -- Current fill area interior style. (Refer to Set Fill Interior Style function.) intout(1) -- Current fill area color index. intout(2) -- Current fill area style index. (Refer to Set Fill Style Index function.) intout(3) -- Current writing mode. (Refer to the Set Writing Mode function.) intout(4) -- Current fill perimeter status.

C BINDING

Procedure Name	vqf_attributes(handle, attrib)
Data Types	WORD vqf_attributes( ); WORD handle; WORD attrib[4];
Input Arguments	handle = contrl[6]
Output Arguments	attrib[0] = intout[0] attrib[1] = intout[1] attrib[2] = intout[2] attrib[3] = intout[3]

INQUIRE CURRENT GRAPHIC TEXT ATTRIBUTES	This function returns the current setting of all attributes that affect graphic text, such as text size, text color, text face alignment, baseline rotation, and writing mode.

Input

contrl(0) --	Opcode = 38.
contrl(1) --	Number of input vertices = 0.
contrl(3) --	Length of intin array = 0.
contrl(6) --	Device handle.

Output

contrl(2) --	Number of output vertices = 2.
contrl(4) --	Length of intout = 6.
intout(0) --	Current graphic text face.
intout(1) --	Current graphic text color index.
intout(2) --	Current angle of rotation of text baseline (in tenths of degrees 0-3600).
intout(3) --	Current horizontal alignment. (Refer to Set Graphic Text Alignment function.)
intout(4) --	Current vertical alignment. (Refer to Set Graphic Text Alignment function.)
intout(5) --	Current writing mode. (Refer to the Set Writing Mode function.)
ptsout(0) --	Current character width in current coordinate system.
ptsout(1) --	Current character height in current coordinate system.
ptsout(2) --	Current character cell width in current coordinate system.
ptsout(3) --	Current character cell height in current coordinate system.

C BINDING

Procedure Name vqt_attributes(handle, attrib)

Data Types

```
WORD vqt_attributes ( );
WORD handle;
WORD attrib[10];
```

Input Arguments handle = contrl[6]

Output Arguments

```
attrib[0] = intout[0]
attrib[1] = intout[1]
  .
  .
  .
attrib[5] = intout[5]
attrib[6] = ptsout[0]
  .
  .
  .
attrib[9] = ptsout[3]
```

INQUIRE TEXT EXTENT

This function returns a rectangle that encloses the requested string. The coordinates of the vertices are given relative to a coordinate system defined such that the extent rectangle touches both the x and y axes, and the string is in the first quadrant. All text attributes, including style and baseline rotation, affect the calculation.

Figure 8-1. Inquire Text Extent Function

Input

contrl(0)	--	Opcode = 116.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Number of words in text.
contrl(6)	--	Device handle.
intin	--	Character string in current character set.

364

Output	contrl(2) --	Number of output vertices = 4.
	contrl(4) --	Length of intout array = 0.
	ptsout(0) --	delta-x for point 1 of the string in the current coordinate system.
	ptsout(1) --	delta-y for point 1 of the string in the current coordinate system.
	ptsout(2) --	delta-x for point 2 of the string in the current coordinate system.
	ptsout(3) --	delta-y for point 2 of the string in the current coordinate system.
	ptsout(4) --	delta-x for point 3 of the string in the current coordinate system.
	ptsout(5) --	delta-y for point 3 of the string in the current coordinate system.
	ptsout(6) --	delta-x for point 4 of the string in the current coordinate system.
	ptsout(7) --	delta-y for point 4 of the string in the current coordinate system.

C BINDING

Procedure Name vqt_extent(handle, string, extent)

Data Types

```
WORD vqt_extent( );
WORD handle;
WORD extent[8];
BYTE string[];
```

Input Arguments

```
handle = contrl[6]
string = intin
```

Output Arguments

```
extent[0] = ptsout[0]
  .
  .
  .
extent[7] = ptsout[7]
```

INQUIRE CHARACTER CELL WIDTH

This function returns the character cell width for a specified character in the current text face. The character cell width is the distance from the left edge of the character to the left edge of the character that follows it in a text string. Special effects and rotation do not apply. GEM VDI returns all values in the current coordinate system.

Figure 8-2. Character Cell Definition

Input

contrl(0)	--	Opcode = 117.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 1.
contrl(6)	--	Device handle.
intin(0)	--	Character value in current character set in ADE format.

Output	contrl(2) --	Number of output vertices = 3.
	contrl(4) --	Length of intout array = 1.
	intout(0) --	ADE value of the character being inquired on; -1 if an invalid character (status).
	ptsout(0) --	Cell width of the character in the current coordinate system.
	ptsout(1) --	0.
	ptsout(2) --	Left character alignment delta.
	ptsout(3) --	0.
	ptsout(4) --	Right character alignment delta.
	ptsout(5) --	0.

C BINDING

Procedure Name

```
status = vqt_width( handle, character,
     &cell_width, &left_delta, &right_delta )
```

Data Types

```
WORD status;
WORD vqt_width( );
WORD handle;
BYTE character;
WORD cell_width;
WORD left_delta;
WORD right_delta;
```

Input Arguments

```
handle = contrl[6]
character = intin[0]
```

Output Arguments

```
status = intout[0]
cell_width = ptsout[0]
left_delta = ptsout[2]
right_delta = ptsout[4]
```

INQUIRE FACE NAME AND INDEX

This function returns a 32-character string that describes the face. The face is selected by its element number (1 to the number of faces available). One word of zero in the intin array terminates the string.

The string describing the face is returned in ADE form in intout(1...32). The face ID to access this face with Set Text Face is returned in intout(1). The first 16 characters name the face. The next 16 characters describe the style and weight. See Table 8-1 for a sample of the possible configurations.

Table 8-1. Face Names and Styles

Face Name	Styles
Swiss 721	Light
Swiss 721	Thin Italic
Dutch 801	Roman
Dutch 801	Bold Italic

Input

contrl(0) -- Opcode = 130.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 1.
contrl(6) -- Device handle.

intin(0) -- Element number.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 33.

intout(0) -- ID number.
intout(1) to
intout(32) - 32 ADE.

C BINDING

Procedure Name	index = vqt_name(handle, element_num, name)
Data Types	WORD index; WORD vqt_name( ); WORD handle; WORD element_num, BYTE name[32];
Input Arguments	handle = contr[6] element_num = intin[0]
Output Arguments	index = intout[0] name[0] = intout[1] . . . name[31] = intout[32]

Note: The BYTE array elements contain the eight least significant bits of the intout array elements. The array is terminated with a null byte.

369

INQUIRE CURRENT FACE INFORMATION

This function returns size information for the current face with the current size and special effects. Because the special effects may change the cell width and extent, a value is returned to allow the use of the width information returned in Inquire Character Cell Width. When the character is skewed, the cell contains left and right offsets as shown in Figure 8-3.

Figure 8-3. Right and Left Offset

Input

contrl(0) -- Opcode = 131.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 5.
contrl(4) -- Length of output array = 2.

intout(0) -- Minimum ADE (ASCII Decimal Equivalent) the first character in this face.
intout(1) -- Maximum ADE, the last character in this face.

ptsout(0) -- Maximum cell width not including special effects.
ptsout(1) -- Bottom line distance relative to baseline.
ptsout(2) -- Special effects delta x. The current special effects increase character width by this amount.

---

ptsout(3) --	Descent line distance relative to baseline.
ptsout(4) --	Left offset; (see Figure 8-2) positive value relative to position.
ptsout(5) --	Half distance relative to baseline.
ptsout(6) --	Right offset (see Figure 8-2).
ptsout(7) --	Ascent distance relative to baseline.
ptsout(8) --	0.
ptsout(9) --	Top distance relative to baseline.

C BINDING

Procedure Name

```
vqt_fontinfo( handle, &minADE, &maxADE,
   distances, &maxwidth, effects )
```

Data Types

```
WORD vqt_fontinfo( );
WORD handle;
WORD minADE;
WORD maxADE;
WORD distances[5];
WORD maxwidth;
WORD effects[3];
```

Input Arguments

```
handle = contrl[6]
```

Output Arguments

```
minADE = intout[0]
maxADE = intout[1]
distances[0] = ptsout[1]
distances[1] = ptsout[3]
distances[2] = ptsout[5]
distances[3] = ptsout[7]
distances[4] = ptsout[9]
maxwidth = ptsout[0]
effects[0] = ptsout[2]
effects[1] = ptsout[4]
effects[2] = ptsout[6]
```

INQUIRE CELL ARRAY

This function returns the cell array definition of the specified pixels. Color indices are returned one row at a time, starting from the top of the rectangular area, proceeding downward.

Note: This function is not required and may not be available on all devices.

Input

contrl(0)	--	Opcode = 27.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 0.
contrl(6)	--	Device handle.
contrl(7)	--	Length of each row in color index array.
contrl(8)	--	Number of rows in color index array.
ptsin(0)	--	x-coordinate of lower left corner in current coordinate system.
ptsin(1)	--	y-coordinate of lower left corner in current coordinate system.
ptsin(2)	--	x-coordinate of upper right corner in current coordinate system.
ptsin(3)	--	y-coordinate of upper right corner in current coordinate system.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of the color index array, same as contrl(3).
contrl(9)	--	Number of elements used in each row of color index array.
contrl(10)	--	Number of rows used in color index array.
contrl(11)	--	Invalid value flag. 0 -- if no errors 1 -- if a color value could not be determined for some pixel

37

intout	--	Color index array, stored one row at time.
		-1 -- indicates that a color index could not be determined for that particular pixel

C BINDING

Procedure Name

```
vq_cellarray( handle, pxyarray, row_length,
    num_rows, &el_used, &rows_used, &status,
    colarray )
```

Data Types

```
WORD vq_cellarray( );
WORD handle;
WORD pxyarray[4];
WORD row_length;
WORD num_rows;
WORD el_used;
WORD rows_used;
WORD status;
WORD colarray[n];
```

Input Arguments

```
handle = contrl[6]
pxyarray[0] = ptsin[0]
pxyarray[1] = ptsin[1]
pxyarray[2] = ptsin[2]
pxyarray[3] = ptsin[3]
row_length = contrl[7]
num_rows = contrl[8]
```

Output Arguments

```
el_used = contrl[9]
rows_used = contrl[10]
status = contrl[11]
colarray[0] = intout[0]
  .
  .
colarray[n] = intin[n]
```

INQUIRE INPUT MODE This function returns the current input mode for the specified logical input device: locator, valuator, choice, and string.

Input

```
contrl(0) --   Opcode = 115.
contrl(1) --   Number of input vertices = 0.
contrl(3) --   Length of intin array = 1.
contrl(6) --   Device handle.

intin(0)  --   Logical input device.

               1 = locator
               2 = valuator
               3 = choice
               4 = string
```

Output

```
contrl(2) --   Number of output vertices = 0.
contrl(4) --   Length of intout array = 1.

intout(0) --   Input mode.

               1 = request
               2 = sample
```

C BINDING

Procedure Name `vqin_mode( handle, dev_type, &input_mode )`

Data Types

```
WORD vqin_mode( );
WORD handle;
WORD dev_type;
WORD input_mode;
```

Input Arguments

```
handle = contrl[6]
dev_type = intin[0]
```

Output Arguments

```
input_mode = intout[0]
```

End of Section 8

Section 9
ESCAPES

ESCAPE

The Escape function allows the application program to access the special capabilities of a graphics device. GEM VDI predefines some escape functions; others can be defined for specific devices. The parameters passed depend on the escape function the application requests.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices.
contrl(3) -- Number of input parameters.
contrl(5) -- Function identifier (id).
contrl(6) -- Device handle.

Table 9-1. Escape Function Identifiers

Number	Description
1	INQUIRE ADDRESSABLE ALPHA CHARACTER CELLS
2	EXIT ALPHA MODE
3	ENTER ALPHA MODE
4	ALPHA CURSOR UP
5	ALPHA CURSOR DOWN
6	ALPHA CURSOR RIGHT
7	ALPHA CURSOR LEFT
8	HOME ALPHA CURSOR
9	ERASE TO END OF ALPHA SCREEN
10	ERASE TO END OF ALPHA TEXT LINE
11	DIRECT ALPHA CURSOR ADDRESS
12	OUTPUT CURSOR ADDRESSABLE ALPHA TEXT
13	REVERSE VIDEO ON

Table 9-1. (continued)

Number	Description
14	REVERSE VIDEO OFF
15	INQUIRE CURRENT ALPHA CURSOR ADDRESS
16	INQUIRE TABLET STATUS
17	HARD COPY
18	PLACE GRAPHIC CURSOR AT LOCATION
19	REMOVE LAST GRAPHIC CURSOR
20	FORM ADVANCE
21	OUTPUT WINDOW
22	CLEAR DISPLAY LIST
23	OUTPUT BIT IMAGE FILE
24-59	UNUSED BUT RESERVED FOR FUTURE EXPANSION
60	SELECT PALETTE
61-90	UNUSED BUT RESERVED FOR FUTURE EXPANSION
91	INQUIRE PALETTE FILM TYPES
92	INQUIRE PALETTE DRIVER STATE
93	SET PALETTE DRIVER STATE
94	SAVE PALETTE DRIVER STATE
95	SUPPRESS PALETTE MESSAGES
96	PALETTE ERROR INQUIRE
98	UPDATE METAFILE EXTENTS
99	WRITE METAFILE ITEM
100	CHANGE GEM VDI FILENAME
>100	UNUSED AND AVAILABLE FOR USE

	intin	--	Function-dependent information described on following pages.
	ptsin	--	Array of input coordinates for escape function.
Output	contrl(2)	--	Number of output vertices.
	contrl(4)	--	Number of output parameters.
	intout	--	Array of output parameters.
	ptsout	--	Array of output coordinates.

ESCAPE 1: INQUIRE ADDRESSABLE ALPHA CHARACTER CELLS	This escape returns information to the calling program about the number of vertical (row) and horizontal (column) positions at which the alpha cursor can be positioned on the screen. Typically, only screens support alpha text.
Input	contrl(0) -- Opcode = 5. contrl(1) -- Number of input vertices = 0. contrl(3) -- Length of intin array = 0. contrl(5) -- Function id = 1. contrl(6) -- Device handle.
Output	contrl(2) -- Number of output vertices = 0. contrl(4) -- Length of intout array = 2. intout(0) -- Number of addressable rows on the screen, (-1 indicates cursor addressing not possible). intout(1) -- Number of addressable columns on the screen, (-1 indicates cursor addressing not possible).

C BINDING

Procedure Name vq_chcells(handle, &rows, &columns)

Data Types

```
WORD vq_chcells ( );
WORD handle;
WORD rows;
WORD columns;
```

Input Arguments handle = contrl[6]

Output Arguments

```
rows = intout[0]
columns = intout[1]
```

ESCAPE 2: EXIT ALPHA MODE	This escape causes the graphics device to enter graphics mode if graphics mode is different from alpha mode. It is used to exit alpha cursor addressing mode explicitly and to make the transition from alpha to graphics mode properly.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 2.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name v_exit_cur(handle)

Data Types

```
WORD v_exit_cur ( );
WORD handle;
```

Input Arguments handle = contrl[6]

ESCAPE 3: ENTER ALPHA MODE

This escape causes the graphics device to exit graphics mode if graphics mode is different from alpha mode. It is used to enter the alpha cursor addressing mode explicitly and to make the transition from graphics to alpha mode properly. This opcode also returns the cursor to the upper left character cell of the display device.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 0.
contrl(5)	--	Function id = 3.
contrl(6)	--	Device handle.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_enter_cur( handle )
```

Data Types

```
WORD v_enter_cur ( );
WORD handle;
```

Input Arguments

```
handle = contrl[6]
```

ESCAPE 4: ALPHA CURSOR UP	This escape moves the alpha cursor up one row without altering its horizontal position. If the cursor is already at the top margin, nothing happens.	
Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 4.
	contrl(6) --	Device handle.
Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_curup(handle)
Data Types	WORD v_curup ( ); WORD handle;
Input Arguments	handle = contrl[6]

ESCAPE 5: ALPHA CURSOR DOWN

This escape moves the alpha cursor down one row without altering its horizontal position. If the cursor is already at the bottom margin, nothing happens.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 5.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name

```
v_curdown( handle )
```

Data Types

```
WORD v_curdown ( );
WORD handle;
```

Input Arguments

```
handle = contrl[6]
```

ESCAPE 6: ALPHA CURSOR RIGHT

The Alpha Cursor Right escape moves the alpha cursor right one column without altering its vertical position. If the cursor is already at the right margin, nothing happens.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 6.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name v_curright(handle)

Data Types

```
WORD v_curright ( );
WORD handle;
```

Input Arguments handle = contrl[6]

ESCAPE 7: ALPHA CURSOR LEFT	The Alpha Cursor Left escape moves the alpha cursor left one column without altering its vertical position. If the cursor is already at the left margin, nothing happens.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 7.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_curleft(handle)
Data Types	WORD v_curleft ( ); WORD handle;
Input Arguments	handle = contrl[6]

ESCAPE 8: HOME ALPHA CURSOR	This escape moves the alpha cursor to the home position, usually the upper left character cell of the display device.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 8.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name v_curhome(handle)

Data Types

```
WORD v_curhome ( );
WORD handle;
```

Input Arguments handle = contrl[6]

ESCAPE 9: ERASE TO END OF ALPHA SCREEN	This escape erases the display surface from the current alpha cursor position to the end of the alpha screen. The current alpha cursor location does not change.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 9.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_eeos(handle)
Data Types	WORD v_eeos ( ); WORD handle;
Input Arguments	handle = contrl[6]

ESCAPE 10: ERASE TO END OF ALPHA TEXT LINE	This escape erases the display surface from the current alpha cursor position to the end of the current alpha text line. The current alpha cursor location does not change.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 10.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name v_eeol(handle)

Data Types

```
WORD v_eeol ( );
WORD handle;
```

Input Arguments handle = contrl[6]

388

ESCAPE 11: DIRECT ALPHA CURSOR ADDRESS

The Direct Alpha Cursor Address escape moves the alpha cursor directly to the specified row and column address anywhere on the display surface. Addresses beyond the displayable range of the screen are set to the nearest value that is within the displayable range of the screen.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 2.
contrl(5) -- Function id = 11.
contrl(6) -- Device handle.

intin(0) -- Row number (1 to maximum number of rows).
intin(1) -- Column number (1 to maximum number of columns).

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name

```
vs_curaddress( handle, row, column )
```

Data Types

```
WORD vs_curaddress ( );
WORD handle;
WORD row;
WORD column;
```

Input Arguments

```
handle = contrl[6]
row = intin[0]
column = intin[1]
```

ESCAPE 12: OUTPUT CURSOR ADDRESSABLE ALPHA TEXT

This escape displays a string of alpha text starting at the current cursor position. The alpha text attributes currently in effect determine alpha text attributes.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Number of characters in character string.
contrl(5)	--	Function id = 12.
contrl(6)	--	Device handle.
intin	--	Text string in ADE.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_curtext( handle, &string )
```

Data Types

```
WORD v_curtext ( );
WORD handle;
BYTE string[];
```

Input Arguments

```
handle = contrl[6]
string = intin
```

Note: The BYTE values contain the eight least significant bits of the intin array.

ESCAPE 13: REVERSE VIDEO ON	This escape displays all subsequent alpha text in reverse video.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 13.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_rvon(handle)
Data Types	WORD r_von ( ); WORD handle;
Input Arguments	handle = contrl[6]

ESCAPE 14: REVERSE VIDEO OFF	This escape displays all subsequent alpha C text in normal video format.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 14.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_rvoff(handle)
Data Types	WORD v_rvoff ( ); WORD handle;
Input Arguments	handle = contrl[6]

392

ESCAPE 15: INQUIRE CURRENT ALPHA CURSOR ADDRESS	This escape returns the current position of the alpha cursor in row, column coordinates.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 15.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 2.
	intout(0) --	Row number (1 to the maximum number of rows).
	intout(1) --	Column number (1 to the maximum number of columns).

C BINDING

Procedure Name	vq_curaddress(handle, &row, &column)
Data Types	WORD vq_curaddress ( ); WORD handle; WORD row; WORD handle;
Input Arguments	handle = contrl[6]
Output Arguments	row = intout[0] column = intout[1]

ESCAPE 16: INQUIRE TABLET STATUS

This escape returns the availability status of a graphics tablet, mouse, joystick, or other similar device.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 16.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Tablet status.

0 = tablet not available
1 = tablet available

C BINDING

Procedure Name

```
status = vq_tabstatus( handle )
```

Data Types

```
WORD vq_tabstatus ( );
WORD handle;
WORD status;
```

Input Arguments

```
handle = contrl[6]
```

Output Arguments

```
status = intout[0]
```


394

ESCAPE 17: HARD COPY	The device generates a hard copy with this escape. The escape is device-specific and copies the physical screen to a printer or other attached hard copy device.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 17.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_hardcopy(handle)
Data Types	WORD v_hardcopy ( ); WORD handle;
Input Arguments	handle = contrl[6]

ESCAPE 18: PLACE GRAPHIC CURSOR AT LOCATION

This escape places a graphic cursor at the specified location. The cursor is usually a cross hair cursor and is of the same type as that used for Input Locator, Request Mode. If sample mode input is supported, the application can use this call to generate the cursor for Input Locator, Sample Mode. In memory-mapped devices, the cursor is drawn in XOR mode so GEM VDI can remove it.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 1.
contrl(3)	--	Length of intin array = 0.
contrl(5)	--	Function id = 18.
contrl(6)	--	Device handle.
ptsin(0)	--	x-coordinate of location to place cursor in current coordinate system.
ptsin(1)	--	y-coordinate of location to place cursor in current coordinate system.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_dspcur( handle, x, y )
```

Data Types

```
WORD v_dspcur ( );
WORD handle;
WORD x, y;
```

Input Arguments

```
handle = contrl[6]
x = ptsin[0]
y = ptsin[1]
```

ESCAPE 19: REMOVE LAST GRAPHIC CURSOR	This escape removes the last graphic cursor placed on the screen.	
Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 19.
	contrl(6) --	Device handle.
Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name v_rmcur(handle)

Data Types

```
WORD v_rmcur ( );
WORD handle;
```

Input Arguments handle = contrl[6]

39.

ESCAPE 20: FORM ADVANCE	This escape is required only for printers. It advances the printer page. This escape can be used instead of invoking a Clear Workstation function if it is desirable to retain the current printer display list while advancing to the next page.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 20.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name v_form_adv(handle)

Data Types

```
WORD v_form_adv( );
WORD handle;
```

Input Arguments handle = contrl[6]

ESCAPE 21: OUTPUT WINDOW

This escape is required only for printers. It allows the application to request that a particular rectangular window of the picture be output to the printer. This escape is similar to the Update Workstation function, except that the rectangular area must be specified.

Note that use of this function does not always guarantee that adjacent pictures will abut. Pictures will abut with a resolution of one printer head height.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 0.
contrl(5)	--	Function id = 21.
contrl(6)	--	Device handle.
ptsin(0)	--	x-coordinate of corner of window to be output in NDC/RC.
ptsin(1)	--	y-coordinate of corner of window to be output in NDC/RC.
ptsin(2)	--	x-coordinate of corner of window, diagonally opposite corner selected in ptsin(0), in NDC/RC.
ptsin(3)	--	y-coordinate of corner of window, diagonally opposite corner selected in ptsin(1), in NDC/RC.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name v_output_window(handle, xyarray)

Data Types

```
WORD v_output_window( );
WORD handle;
WORD xyarray[4];
```

Input Arguments

```
handle = contrl[6]
xyarray[0] = ptsin[0]
  .
  .
  .
xyarray[3] = ptsin[3]
```

ESCAPE 22: CLEAR DISPLAY LIST	This escape is required only for printers. It allows the application to request that the printer display list be cleared. It is similar to the Clear Workstation function, but does not cause a form advance on the printer.

Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of intin array = 0.
	contrl(5) --	Function id = 22.
	contrl(6) --	Device handle.

Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name	v_clear_disp_list(handle)
Data Types	WORD v_clear_disp_list( ); WORD handle;
Input Arguments	handle = contrl[6]

401

ESCAPE 23: OUTPUT BIT IMAGE FILE

This escape is required only for printers. It allows the application to request processing of a bit image file (see Appendix I, "Bit Image File Format"). As input parameters, the application provides a filename and information on image transformation and page placement.

The application uses three parameters to control image transformation:

- pixel aspect ratio flag
- x-axis scaling flag
- y-axis scaling flag

The application can set the pixel aspect ratio flag to preserve or ignore the pixel aspect ratio defined in the bit image file. Preserving pixel aspect ratio means the printed object will have the same aspect ratio it had on the device on which it was originally drawn. For example, squares remain squares, and circles remain circles. Ignoring pixel aspect ratio means the printed object will not necessarily have the same aspect ratio it had on the original device.

The application can set the two axis scaling flags independently of each other. The flags determine if the bit image's x or y axes are to be scaled fractionally or in integer multiples. The upward boundary of this scaling is an application-defined rectangle.

If an axis of the bit image is scaled fractionally, it will exactly fit the corresponding axis of the scaling rectangle, with the exception noted below.

If an axis of the bit image is scaled in integer multiples, it might not exactly fit the corresponding axis of the scaling rectangle.

If the scaled bit image does not exactly fit the scaling rectangle, the application can use alignment parameters to locate the bit image within the rectangle. These parameters allow any combination of three vertical and three horizontal positions.

2101

Note: The scaled bit image always resides within the scaling rectangle. If a combination of preserved pixel aspect ratio, scaling, or alignment causes the scaled bit image to extend beyond an edge of the scaling rectangle, GEM VDI clips the bit image to that edge.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 2.

contrl(3) -- Length of intin array = length of filename + 5.
contrl(5) -- Function id = 23.
contrl(6) -- Device handle.

intin(0) -- Aspect ratio flag.

0 = ignore aspect ratio
1 = honor pixel aspect ratio

intin(1) -- Scaling for x-axis.

0 = fractional scaling
1 = integer scaling

intin(2) -- Scaling for y-axis.

0 = fractional scaling
1 = integer scaling

intin(3) -- Horizontal alignment.

0 = left
1 = center
2 = right

intin(4) -- Vertical alignment.

0 = top
1 = middle
2 = bottom

intin(5) -- First character of filename.

.
.
.

intin(n+4)-- Last (nth) character of filename.

40:

	ptsin(0) --	Upper left x (if specified).
	ptsin(1) --	Upper left y (if specified).
	ptsin(2) --	Lower right x (if specified).
	ptsin(3) --	Lower right y (if specified).
Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 0.

C BINDING

Procedure Name

```
v_bit_image( handle, filename, aspect,
             x_scale, y_scale, h_align,
             v_align, xyarray )
```

Data Types

```
WORD v_bit_image();
BYTE filename[];
WORD handle, aspect, x_scale, y_scale,
     h_align, v_align;
WORD xyarray[];
```

Input Arguments

```
handle = contrl[6]
filename = intin[2] . . . intin[n + 1]
aspect = intin[0]
x_scale = intin[1]
y_scale = intin[2]
h_align = intin[3]
v_align = intin[4]
xyarray[0] = ptsin[0]
xyarray[1] = ptsin[1]
xyarray[2] = ptsin[2]
xyarray[3] = ptsin[3]
```

Note: Bytes for the filename array are mapped into the corresponding eight least significant bits of intin. The string must be null-terminated.

403

ESCAPE 60: SELECT PALETTE	This escape allows the selection of the palette on the IBM..medium-resolution color screen.	
Input	contrl(0) --	Opcode = 5.
	contrl(1) --	Number of input vertices = 0.
	contrl(3) --	Length of input array = 1.
	contrl(5) --	Function id = 60.
	contrl(6) --	Device handle.
	intin(0) --	Color selection. 0 = use red, green, brown palette (default) 1 = use cyan, magenta, white palette
Output	contrl(2) --	Number of output vertices = 0.
	contrl(4) --	Length of intout array = 1.
	intout(0) --	Palette selected.

C BINDING

Procedure Name selected = vs_palette(handle, palette)

Data Types

```
WORD vs_palette( );
WORD handle;
WORD palette;
```

Input Arguments

handle = contrl[6]
palette = intin[0]

Output Arguments

selected = intout[0]

404

POLAROID..PALETTE

Use these escapes to modify the operation of the Polaroid Palette image recorder. While their use is not mandatory, they allow construction of a more efficient user interface.

Palette Driver

These escapes affect a header in the palette driver. The header contains information on the current state of the driver and the types of films it can use. The palette driver contains exposure tables for five film types. A 25-character string describes each film type, stating its manufacturer and its ASA number. These strings are padded with blanks if the information requires less than 25 characters.

Seventy-two colors are defined for each film type. These colors are mapped to an 8 x 9 array with ASCII capitals (A...H), naming the columns and ASCII digits (1...9), numbering the rows. A color is selected by its letter and number. For example, A2 identifies the second color in column A.

Numbers also identify the port to which the palette is connected, an f-stop control, and a resolution control for environments where memory size prevents the use of the Palette's full capabilities.

The palette driver normally outputs its messages directly to the screen. These messages include error messages and user prompts.

Error Messages

The palette error messages appear when the application calls GEM VDI with a function other than Open Workstation, Close Workstation, or any of the Escape functions. These messages can be suppressed with Escape 95. The application can then use the code returned from Escape 96 to inform the user of the error condition.

ESCAPE 91: INQUIRE PALETTE FILM TYPES

This escape returns five strings that describe the films that the driver is currently capable of exposing. The strings are padded with spaces if they have fewer than 25 characters. The strings are returned as ADE integers in intout.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 0.
contrl(5)	--	Function id = 91.
contrl(6)	--	Device handle.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout = 125.
intout	--	5 sets of 25 ADE character strings.

C BINDING

Procedure Name

```
vqp_films( handle, film_names )
```

Data Types

```
WORD vqp_films( );
WORD handle;
WORD film_names[125];
```

Input Arguments

```
handle = contrl[6]
```

Output Arguments

```
film_names = intout
```

Note: Intout words (ADE) are converted to byte string.

ESCAPE 92: INQUIRE PALETTE DRIVER STATE

This escape returns a block of data that describes the current state of the driver. The state can be updated by changing this block and returning it to the driver with Escape 93.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin = 0.
contrl(5) -- Function id = 92.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 20.

intout(0) -- Port number.

0 = first comm port

intout(1) -- Film number (0...$).
intout(2) -- Lightness control (-3...3).

Each integer increase represents opening the aperture 1/3 of an f-stop. A -3 results in an exposure half as long as normal, while a 3 doubles the exposure time.

intout(3) -- Interlace flag.

0 = noninterlaced
1 = interlaced

A noninterlaced picture requires slightly more than half the memory of an interlaced picture.

intout(4) -- Planes, a number (1...4) corresponding to number of colors (2...16).

intout(5 to 20) -- Two-character color codes for 8-color indices stored in ADE format.

407

C BINDING

Procedure Name

```
vqp_state( handle, &port, &film_name,
    &lightness, &interlace, &planes,
    &indexes )
```

Data Types

```
WORD vqp_state( );
WORD handle;
WORD port;
WORD film_name;
WORD lightness;
WORD interlace;
WORD planes;
WORD indexes[8][2];
```

Input Arguments

```
handle = contrl[6]
```

Output Arguments

```
port = intout[0]
film_name = intout[1]
lightness = intout[2]
interlace = intout[3]
planes = intout[4]
indexes = intout[5...20]
```

ESCAPE 93: SET PALETTE DRIVER STATE	This escape moves a block of characteristics into the driver. Use this function after ESCAPE 92.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 20.
contrl(5)	--	Function id = 93.
contrl(6)	--	Device handle.
intin(0)	--	Port number. 0 = first comm port
intin(1)	--	Film number (0...4).
intin(2)	--	Lightness control (-3...3). Each integer indicates opening the aperture 1/3 an f-stop. A -3 results in an exposure half as long as normal, while a 3 doubles the exposure time.
intin(3)	--	Interlace flag. 0 = noninterlaced 1 = interlaced
intin(4)	--	Planes (1 to 4), number corresponds to number of colors (2 to 16).
intin(5 to 20)	--	Color codes for up to 16 colors.

C BINDING

Procedure Name	vsp_state(handle, port, film_num, lightness, interlace, planes, indexes)
Data Types	WORD vsp_style( ); WORD handle; WORD port; WORD film_num; WORD lightness, WORD interlace; WORD planes; WORD indexes[8][2];
Input Arguments	handle = contrl[6] port = intin[0] film_num = intin[1] lightness = intin[2] interlace = intin[3] planes = intin[4] indexes = intin[5-20]

410

ESCAPE 94: SAVE PALETTE DRIVER STATE	This escape saves the current state of the driver to disk. The application can change the default film and index mapping with this escape.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 94.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 0.

C BINDING

Procedure Name vsp_save(handle)

Data Types

```
WORD vsp_save( );
WORD handle;
```

Input Arguments handle = contrl[6]

411

ESCAPE 95: SUPPRESS PALETTE MESSAGES

This escape allows the application to suppress the messages the palette driver normally outputs to the screen. These messages are either error messages or user prompts. Refer to Escape 96 for the messages and their codes.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 95.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of output array = 0.

C BINDING

Procedure Name vsp_message(handle)

Data Types

```
WORD vsp_message( );
WORD handle;
```

Input Arguments handle = contrl[6]

412

ESCAPE 96: PALETTE ERROR INQUIRE

This escape returns an error code so the application can notify the user of a problem. This escape also returns codes for pending user prompts. The error is not cleared, so a message can be displayed if such messages are not suppressed.

Input

contrl(0) -- Opcode = 5.
contrl(1) -- Number of input vertices = 0.
contrl(3) -- Length of intin array = 0.
contrl(5) -- Function id = 96.
contrl(6) -- Device handle.

Output

contrl(2) -- Number of output vertices = 0.
contrl(4) -- Length of intout array = 1.

intout(0) -- Error codes and pending user prompts.

0 = no error
1 = open dark slide for print film
2 = no port at location specified in driver
3 = palette not found at specified port
4 = video cable disconnected
5 = operating system does not allow memory allocation
6 = not enough memory to allocate buffer
7 = memory not deallocated
8 = driver file not found
9 = driver file found is not correct type
10= prompt user to process print film

C BINDING

Procedure Name	status = vqp_error(handle)
Data Types	WORD vqp_error( ); WORD handle;
Output Arguments	status = intout[0]
Input Arguments	handle = contrl[6]

414

ESCAPE 98: UPDATE METAFILE EXTENTS	The values passed in the ptsin array are used to update the extents information in the metafile header. The extents information may be used by some applications to provide a quick indication of the minimum rectangle which will bound all primitives output to the metafile. If the Update Metafile Extents escape is not used when outputting to the metafile, zeroes will be written in the extents information portion of the metafile header.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 2.
contrl(3)	--	Length of intin array = 0.
contrl(5)	--	Function id = 98.
contrl(6)	--	Device handle.
ptsin(0)	--	Minimum x value of the minimum bounding rectangle.
ptsin(1)	--	Minimum y value of the minimum bounding rectangle.
ptsin(2)	--	Maximum x value of the minimum bounding rectangle.
ptsin(3)	--	Maximum y value of the minimum bounding rectangle.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
v_meta_extents(handle, min_x, min_y,
      max_x, max_y)
```

Data Types

```
WORD v_meta_extents( );
WORD handle, min_x, min_y, max_x, max_y;
```

Input Arguments

```
handle = contrl[6];
min_x = ptsin[0];
min_y = ptsin[1];
max_x = ptsin[2];
max_y = ptsin[3];
```

ESCAPE 99: WRITE METAFILE ITEM

The parameters passed in the intin and ptsin arrays are written to the metafile with an opcode defining the item as a user-defined metafile item. Intin(0) should contain a sub-opcode that defines what type of user-defined metafile item is being written. Sub-opcodes numbered 0 through 100 are reserved; the sub-opcode you use to define your metafile item should be numbered 101 or higher.

Input

```
contrl(0) --   Opcode = 5.
contrl(1) --   Number of input vertices.
contrl(3) --   Length of intin array.
contrl(5) --   Function id = 99.
contrl(6) --   Device handle.

intin     --   User-defined information.
intin(0)  --   Sub-opcode.

ptsin     --   User-defined information.
```

Output

```
contrl(2) --   Number of output vertices = 0.
contrl(4) --   Length of intout array = 0.
```

C BINDING

Procedure Name

```
v_write_meta(handle, num_intin, intin,
    num_ptsin, ptsin)
```

Data Types

```
WORD v_write_meta();
WORD handle, num_intin, num_ptsin;
WORD intin[num_intin], ptsin[num_ptsin];
```

Input Arguments

```
handle = contrl[6];
num_intin = contrl[3];
num_ptsin = contrl[1];
intin = intin;
ptsin = ptsin;
```

ESCAPE 100: CHANGE GEM VDI FILENAME

This escape renames a metafile from GEMFILE.GEM to the specified name and maintains the file extension .GEM. A path name and drive can be specified to locate the file somewhere other than on the current drive and directory. Contrl(3) contains the length of the file specification string.

Note: This escape must be called immediately after Open Workstation, or it has no effect. It also closes any open metafiles.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = number of significant characters [1...74].
contrl(5)	--	Function id = 100.
contrl(6)	--	Device handle.
intin(0 to n)	--	Path/filename.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

C BINDING

Procedure Name

```
vm_filename( handle, filename )
```

Data Types

```
WORD vm_filename( );
WORD handle;
BYTE filename[ ];
```

Input Arguments

```
handle = contrl[6]
filename = intin[0-n]
```

Note: The filename must be null-terminated.

End of Section 9

Command line syntax error

Description: The GEM VDI command line includes an illegal character, path, or drive id.

Solution: Check for conformance to your operating system's conventions for specifying command lines. Reenter the command line after correcting illegal entries.

Unable to find ASSIGN.SYS

Description: This message appears when GEM VDI is unable to find the ASSIGN.SYS file in the specified location.

Solution: Locate the ASSIGN.SYS file, checking drives and specific directories and subdirectories. Reenter the command with the correct location.

Error reading ASSIGN.SYS

Description: The format of the ASSIGN.SYS file is incorrect. GEM VDI cannot use the file.

Solution: Refer to Appendix B for the correct format for the ASSIGN.SYS file.

Memory table corrupted

Description: This message appears when memory is corrupted.

Solution: Reboot your system.

Insufficient memory

Description: This message appears when you try to reserve memory and not enough memory exists for allocation.

Solution: If your system has adequate memory to run GEM VDI, reboot your system.

Invalid memory block address

Description: This message occurs when the memory is corrupted.

Solution: Reboot the system.

Drive specification not allowed in ASSIGN.SYS

Description: This error appears when you specify a drive id in the ASSIGN.SYS file, which is illegal.

Solution: Remove the drive id from the file with your text editor. Refer to Appendix B for the correct format of an ASSIGN.SYS file.

Illegal device id in ASSIGN.SYS

Description: This error appears when the device id number is greater than 32767 or an alphanumeric string, for example 12D4.

Solution: Refer to Table 1-1 in Section 1 for the correct numbers to assign to devices, and correct the ASSIGN.SYS file with your text editor.

Partial record found in ASSIGN.SYS

Description: This error appears when a partial ASSIGN.SYS entry exists.

Solution: Check your ASSIGN.SYS file for incomplete device id numbers or filenames. Refer to Appendix B for the correct ASSIGN.SYS file format.

Invalid filename encountered in ASSIGN.SYS

Description: This error appears when a filename in the ASSIGN.SYS file is too long or contains illegal characters.

Solution: Refer to Appendix B for the ASSIGN.SYS file-naming conventions.

Requested path not found

Description: This message appears when GEM VDI does not find the requested path specifying the locations of the device drivers.

Solution: Respecify the path with the correct path name.

ASSIGN.SYS file is empty

Description: This message appears when GEM VDI finds an empty ASSIGN.SYS file.

Solution: Enter the necessary information with your text editor. Refer to Appendix B for the necessary ASSIGN.SYS file contents.

Driver file not found

Description: GEM VDI cannot find the first driver specified in the ASSIGN.SYS file.

Solution: Make sure that the driver is in the specified drive, in the correct directory, and in the correct subdirectory.

Corrupted driver file

Description: GEM VDI finds the device driver, but is unable to use it.

Solution: Use your distribution disk to make another copy of the device driver. Try to use the new copy. Contact your dealer if the device driver is unusable.

End of Appendix A

REQUIREMENTS

The ASSIGN.SYS file is parsed by the GDOS to create the assignment table. The assignment table resides in memory and is referenced when the application makes an Open Workstation call. The information required by the ASSIGN.SYS includes the device id number and the device driver filename and corresponding faces.

Device Id Numbers

Table B-1. Device Id Numbers

Type	Number
Monitor	1-10
Plotter	11-20
Printer	21-30
Metafile	31-40
Camera	41-50
Tablet	51-60

Device Driver Filename

The device driver filenames follow specific naming conventions:

- o They must have eight or fewer characters.
- o The first character must be alphabetic.
- o The file extension must be .SYS.

FORMAT

Figure B-1 shows the ASSIGN.SYS file format:

Device Id	Driver Filename	Face Name
01	SCREEN.SYS	FACE1.FNT

Figure B-1. ASSIGN.SYS File Format

422

SAMPLE ASSIGN.SYS

```
 21 printer.fnt
;comments, if desired
face1.fnt   ;face1 description
face2.fnt   ;face2 description
face3.fnt   ;face3 description
01 screen.fnt
;comments, if desired
face4.fnt   ;face4 description
face5.fnt   ;face5 description
11 plotter.fnt
;comments, if desired
face6.fnt   ;face6 description
face7.fnt   ;face7 description
```

End of Appendix B

Appendix C
GEM VDI METAFILE FORMAT

INTRODUCTION

The metafile driver outputs the information specified below and performs the described operations for the indicated opcodes.

STANDARD METAFILE ITEM FORMAT

Most function requests passed to the metafile driver result in a standard format metafile item being written to the metafile buffer. In a standard format metafile item, the control, integer, and vertex parameters are written to the metafile in the following format:

word	value	description
0	contrl[0]	opcode
1	contrl[1]	vertex count
2	contrl[3]	integer parameter count
3	contrl[5]	sub-opcode (or zero)
4...	ptsin[0-n]	vertices (if provided)
n+4...	intin[0-m]	integer parameters (if provided)

Note that nothing will be output for the ptsin or intin information if the vertex count or the integer parameter count is zero.

The following function requests result in the output of a standard metafile item:

3	clear workstation
4	update workstation
5, 2	exit alpha mode escape
5, 3	enter alpha mode escape
5,21	advance form
5,21	output window
5,22	clear display list
5,23	output bit image file
6	polyline
7	polymarker
8	text
9	fill area
11, 1	bar
11, 2	arc

11, 3	pie
11, 4	circle
11, 5	ellipse
11, 6	elliptical arc
11, 7	elliptical pie
11, 8	rounded rectangle
11, 9	filled rounded rectangle
11,10	justified graphics text
12	set character height, absolute mode
13	set character baseline vector
14	set color representation
15	set polyline linetype
16	set polyline line width
17	set polyline color index
18	set polymarker type
19	set polymarker height
20	set polymarker color index
21	set text face
22	set text color index
23	set fill interior style
24	set fill style index
25	set fill color index
32	set writing mode
39	set graphic text alignment
104	set fill perimeter visibility
106	set graphic text special effects
107	set character height, points mode
108	set polyline end styles
112	set user-defined fill pattern
113	set user-defined line style pattern
114	fill rectangle
129	set clipping

NONSTANDARD METAFILE ITEMS

1 open workstation

The metafile file buffer is initialized and the metafile header is output to it. The workstation description values normally returned by an "open workstation" invocation are returned.

Metafile header format:

word	description
0	0ffffh
1	Length of header in words.
2	100*major version number + minor version number.
3	NDC/RC transformation mode flag 0 = positive y values ascend from origin (origin in lower left corner) 2 = positive y values descend from origin (origin in upper left corner)
4 - 7	Minimum and maximum x and y extent values for the information contained in the metafile. If undefined by the application (see "Escape 98: Update Metafile Extents"), all four values are zero. The values are stored in the following order: minimum x, minimum y, maximum x, maximum y.
8 - 9	Physical page size: page width in tenths of millimeters, followed by page height in tenths of millimeters. If undefined by the application, both values are zero. (See Appendix H, "Reserved Metafile Sub-opcodes.")
10 - 13	The coordinate window which defines the coordinate system used in the metafile. If undefined by the application, all four values are zero. The values are stored in the following order: lower left x, lower left y, upper right x, upper right y. (See Appendix H, "Reserved Metafile Sub-opcodes.")

226

2 close workstation	An end-of-metafile opcode is appended to the metafile file buffer. The metafile file buffer is flushed and the metafile is closed.

End-of-metafile format:

word	description
1	0ffffh

SPECIAL METAFILE ESCAPES

5, 98 update metafile extents	The extents information in the metafile header is updated to indicate the extents passed in the ptsin array.
5, 99 write metafile item escape	A standard format metafile item is written. The first word of the intin array should contain a sub-opcode that can be used by an application to identify the metafile item when it is read in.
5, 100 change GEM VDI filename escape	If any information currently exists in the metafile or metafile buffer, the buffer is flushed and the file is closed. The metafile buffer is reinitialized and rudimentary file name validation is performed. If the drive, path, and filename are valid, they are used to update the file control block (FCB) of the metafile. The metafile will not actually be opened until the first buffer needs to be flushed.

427

INQUIRY FUNCTIONS

5, 1 inquire addressable alpha character cells escape	-1 is returned in both INTOUT parameters to indicate that cursor addressing is not possible.
26 inquire color representation	-1 is returned for the color index to indicate that no value is available.
35 inquire current polyline attributes	The set values are returned.
36 inquire current polymarker attributes	
37 inquire current fill area attributes	
38 inquire current graphic text attributes	
102 extended inquire function	The appropriate inquiry values are returned.
117 inquire character cell width	
131 inquire current face information	

End of Appendix C

GEM VDI defines a standard keyboard so applications can take advantage of special keys not defined in the standard, 7-bit ASCII character set. A 16-bit value is used to return these characters. The high byte contains a binary value assigned to each key. The low byte contains the 7-bit ASCII value, if such a value is defined, or a zero if the code is an extended code.

Table D-1. GEM VDI Standard Keyboard Assignments

High Byte	Low Byte	Character
03	00	CNTL 2 (Nul)
1E	01	CNTL A
30	02	CNTL B
2E	03	CNTL C
20	04	CNTL D
12	05	CNTL E
21	06	CNTL F
22	07	CNTL G
23	08	CNTL H
17	09	CNTL I
24	0A	CNTL J
25	0B	CNTL K
26	0C	CNTL L
32	0D	CNTL M
31	0E	CNTL N
18	0F	CNTL O
19	10	CNTL P
10	11	CNTL Q
13	12	CNTL R
1F	13	CNTL S
14	14	CNTL T
16	15	CNTL U
2F	16	CNTL V
11	17	CNTL W
2D	18	CNTL X
15	19	CNTL Y
2C	1A	CNTL Z
1A	1B	CNTL [
2B	1C	CNTL \
1B	1D	CNTL]
07	1E	CNTL 6
0C	1F	CNTL -
39	20	Space

Table D-1. (continued)

High Byte	Low Byte	Character
02	21	!
28	22	"
04	23	#
05	24	$
06	25	%
08	26	&
28	27	'
0A	28	(
0B	29	)
09	2A	*
0D	2B	+
33	2C	,
0C	2D	-
34	2E	.
35	2F	/
0B	30	0
02	31	1
03	32	2
04	33	3
05	34	4
06	35	5
07	36	6
08	37	7
09	38	8
0A	39	9
27	3A	:
27	3B	;
33	3C	<
0D	3D	=
34	3E	>
35	3F	?
03	40	@
1E	41	A
30	42	B
2E	43	C
20	44	D
12	45	E
21	46	F
22	47	G
23	48	H
17	49	I
24	4A	J
25	4B	K
26	4C	L
32	4D	M
31	4E	N
18	4F	O

Table D-1. (continued)

High Byte	Low Byte	Character
19	50	P
10	51	Q
13	52	R
1F	53	S
14	54	T
16	55	U
2F	56	V
11	57	W
2D	58	X
15	59	Y
2C	5A	Z
1A	5B	[
2B	5C	\
1B	5D	]
07	5E	^
0C	5F	Underscore
29	60	`
1E	61	a
30	62	b
2E	63	c
20	64	d
12	65	e
21	66	f
22	67	g
23	68	h
17	69	i
24	6A	j
25	6B	k
26	6C	l
32	6D	m
31	6E	n
18	6F	o
19	70	p
10	71	q
13	72	r
1F	73	s
14	74	t
16	75	u
2F	76	v
11	77	w
2D	78	x
15	79	y
2C	7A	z
1A	7B	{
2B	7C	|
1B	7D	}
29	7E	~
0E	7F	Rubout (DEL)

Table D-1. (continued)

High Byte	Low Byte	Character
81	00	Alt 0
78	00	Alt 1
79	00	Alt 2
7A	00	Alt 3
7B	00	Alt 4
7B	00	Alt 5
7D	00	Alt 6
7E	00	Alt 7
7F	00	Alt 8
80	00	Alt 9
1E	00	Alt A
30	00	Alt B
2E	00	Alt C
20	00	Alt D
12	00	Alt E
21	00	Alt F
22	00	Alt G
23	00	Alt H
17	00	Alt I
24	00	Alt J
25	00	Alt K
26	00	Alt L
32	00	Alt M
31	00	Alt N
18	00	Alt O
19	00	Alt P
10	00	Alt Q
13	00	Alt R
1F	00	Alt S
14	00	Alt T
16	00	Alt U
2F	00	Alt V
11	00	Alt W
2D	00	Alt X
15	00	Alt Y
2C	00	Alt Z
3B	00	F1
3C	00	F2
3D	00	F3
3E	00	F4
3F	00	F5
40	00	F6
41	00	F7
42	00	F8
43	00	F9
44	00	F10
54	00	F11

Table D-1. (continued)

High Byte	Low Byte	Character
55	00	F12
56	00	F13
57	00	F14
58	00	F15
59	00	F16
5A	00	F17
5B	00	F18
5C	00	F19
5D	00	F20
5E	00	F21
5F	00	F22
60	00	F23
61	00	F24
62	00	F25
63	00	F26
64	00	F27
65	00	F28
66	00	F29
67	00	F30
68	00	F31
69	00	F32
6A	00	F33
6B	00	F34
6C	00	F35
6D	00	F36
6E	00	F37
6F	00	F38
70	00	F39
71	00	F40
73	00	Ctrl left-arrow
4D	00	right-arrow
4D	36	Shift right-arrow
74	00	Ctrl right-arrow
50	00	down-arrow
50	32	Shift down-arrow
48	00	up-arrow
48	38	Shift up-arrow
51	00	Page down
51	33	Shift Page down
76	00	Ctrl Page down
49	00	Page up
49	39	Shift Page up
84	00	Ctrl Page up
77	00	Ctrl Home

Table D-1. (continued)

High Byte	Low Byte	Character
47	00	Home
47	37	Shift Home
52	00	Insert
52	30	Shift Insert
53	00	Delete
53	2E	Shift Delete
72	00	Ctrl Print Screen
37	2A	Print Screen
01	1B	Escape
0E	08	Backspace
82	00	Alt -
83	00	Alt =
1C	0D	CR
1C	0A	Ctrl CR
4C	35	Shift Num Pad 5
4A	2B	Num Pad -
4E	2B	Num Pad +
0F	09	Tab
0F	00	Backtab
4B	00	left-arrow
4B	34	Shift left-arrow
4F	00	End
4F	31	Shift End
75	00	Ctrl End

End of Appendix D

Appendix E
PROCESSOR-SPECIFIC DATA

8086-SPECIFIC DATA

Registers and Interrupts

The address of the Parameter Block is passed in two 16-bit registers (Ds:Dx for the 8086) from the application program to GEM VDI. Pass 0473h in the Cx register. The interrupt is EF.

Note: GEM VDI supports Concurrent..operating systems that support DOS calls of versions 2.0 and above.

Exchange Mouse Movement Vector

For 8086-based microcomputers, the application-dependent code is invoked via a CALL FAR (CALLF) instruction. On entry, the Bx register contains the new x position of the mouse. The Cx register contains the new y position of the mouse. When complete, the application-dependent code should do a RETURN FAR (RETF) instruction with the x,y position of the mouse the driver is to store in Bx, Cx.

Exchange Button Change Vector

For 8086-based processors, the application code is invoked via a CALL FAR (CALLF) instruction with Ax containing the mouse button keys. Keys are encoded by the same rules that apply to the Sample Mouse Button State function. When complete, the application-dependent code should do a RETURN FAR (RETF) instruction with the mouse button state the driver is to store in Ax.

Exchange Cursor Change Vector

For 8086-based machines, the application-dependent code is invoked with a CALL FAR (CALLF) instruction. Upon entry, the Bx register contains the x position and the Cx register the y position. If the application-dependent code does not draw its own cursor, a CALL FAR should be performed to the address returned in contrl(9) and contrl(10) with the x,y position at which to draw the cursor in Bx, Cx. This causes GEM VDI to draw a cursor. When complete, the application should perform a RETURN FAR (RETF) instruction.

Exchange Timer Interrupt Vector

For 8086-based processors, the application-dependent code is invoked with a CALL FAR (CALLF) instruction. When complete, the application should perform a RETURN FAR (RETF) instruction.

68000-SPECIFIC DATA

Registers and Interrupts

The address of the Parameter Block is passed in one 32-bit register, D0.l for 68K from the application program to GEM VDI. D1.w contains the function code 115.

For CP/M-68K, GEM VDI is invoked via TRAP 2. For other 68K operating systems that support GEM VDI, the TRAP is identified in the operating system's manual.

Exchange Mouse Movement Vector

For 68000-based microcomputers, the application-dependent code is invoked via a JUMP TO SUBROUTINE (JSR) instruction. On entry, the D0.w register contains the new x position of the mouse. The D1.w register contains the new y position of the mouse. When complete, the application-dependent code should do a RETURN FROM SUBROUTINE (RTS) instruction with the x,y position of the mouse the driver is to store in D0.w, D1.w.

Exchange Button Change Vector

For 68000-based processors, the application code is invoked via a JUMP TO SUBROUTINE (JSR) instruction with D0.w containing the mouse button keys. Keys are encoded by the same rules that apply to the Sample Mouse Button State function. When complete, the application-dependent code should do a RETURN FROM SUBROUTINE (RTS) instruction with the mouse button state the driver should store in D0.w.

Exchange Cursor Change Vector

For 68000-based machines, the application-dependent code is invoked with a JUMP TO SUBROUTINE (JSR) instruction. Upon entry, the D0.w register contains the x position and the D1.w register the y position. If the application-dependent code does not draw its own cursor, a JUMP TO SUBROUTINE (JSR) instruction should be performed to the address returned in contrl(9) and contrl(10) with the x,y position at which to draw the cursor in D0.w and D1.w. This causes GEM VDI to draw a cursor. When complete, the application should perform a RETURN FROM SUBROUTINE (RTS) instruction.

Exchange Timer Interrupt Vector

For 68000-based processors, the application-dependent code is invoked with a JUMP TO SUBROUTINE (JSR) instruction. When complete, the application should perform a RETURN FROM SUBROUTINE (RTS) instruction.

End of Appendix E

Appendix F
CHARACTER SETS

The system fonts provided with GEM VDI are illustrated in Figure F-1 and F-2. Figure F-1 shows the USASCII character set. Figure F-2 shows the additional characters included to form the international character set.

Note that external fonts (those which are dynamically loaded) do not include characters for decimal equivalents 0 through 31.

Figure F-1. GEM VDI USASCII Character Set

Figure F-2. GEM VDI International Character Set Extension

End of Appendix F

Appendix G
FONT FORMAT

INTRODUCTION

The system fonts and external fonts used in GEM VDI are composed of four parts: the font data, a font header, a character offset table, and a horizontal offset table.

FONT DATA

The font data is organized as a single raster area. The area's height equals the font height and its width equals the sum of the character widths.

The top scan line of the first character in the font is aligned to a byte boundary. The top scan line of the second character is abutted to the first character and is not necessarily byte-aligned. That is, the end of any character and the beginning of the following character often occur within the same byte; no byte alignment occurs within the font form.

Bit padding occurs only at the end of a scan line. Each scan line in the font form begins on a word boundary. The number of bytes from the beginning of one scan line to the beginning of the next is called the form width. The number of scan lines required to draw any character is called the form height.

A flag within the font header indicates the orientation of bytes within a word in the font data. If the flag is cleared, the font data is in a format such that the low byte of a word occurs in memory before the high byte (Intel..format). If the flag is set, the high byte precedes the low byte in memory.

FONT HEADER

The font header contains information that describes global aspects of the font. For example, the name of the face, the font size, the minimum and maximum characters in the font, and any other data that applies to every character of the font are global aspects of that font. The format of the font header is shown in Table G-1.

Table G-1. Font Header Format

Byte Number	Description
0 - 1	face identifier (see the Set Text Face function)
2 - 3	font size in points
4 - 35	face name (see the Inquire Face Name and Index function)
36 - 37	lowest ADE value in the face
38 - 39	highest ADE value in the face
40 - 41	*top line distance
42 - 43	*ascent line distance
44 - 45	*half line distance
46 - 47	*descent line distance
48 - 49	*bottom line distance
50 - 51	width of the widest character in the font
52 - 53	width of the widest character cell in the face
54 - 55	left offset (see the Inquire Current Face Information function)
56 - 57	right offset (see the Inquire Current Face Information function)
58 - 59	thickening: the number of pixels by which to widen thickened characters
60 - 61	underline size: the width (in pixels) of the underline

443

Table G-1. (continued)

Byte Number	Description
62 - 63	lightening mask: the mask used to drop pixels out when lightening; usually 5555H
64 - 65	skewing mask: the mask that is rotated to determine when to perform additional rotation on the character to perform skewing; usually 5555H
66 - 67	flags: bit 0 set if default system font bit 1 set if horizontal offset tables should be used bit 2 byte-swap flag (see "Font Data") bit 3 set if mono-spaced font
68 - 71	pointer to the horizontal offset table
72 - 75	pointer to the character offset table
76 - 79	pointer to the font data
80 - 81	form width (see "Font Data")
82 - 83	form height (see "Font Data")
84 - 87	pointer to the next font (set by the driver)

* - Distances are measured relative to the character baseline and are always a positive value (magnitude rather than offset).

CHARACTER OFFSET TABLE

The character offset table is used to index into the font data and to determine the width of specific characters in the font. It is indexed by relative character value (the ADE value of the desired character, minus the lowest ADE value in the font) and yields the offset from the base of the font data to the beginning of the character definition. The difference between the offset to a character and the offset to the following character gives the width of the character. Note that the character offset table includes one more entry than the number of characters in the font so that a width may be obtained for the final character in the font.

Note: The character offset table is required even for mono-spaced fonts.

HORIZONTAL OFFSET TABLE

The horizontal offset table is indexed by relative character value and yields any additional positive or negative spacing necessary before outputting the character. The horizontal offset table often does not exist. Whether it exists or not is indicated by the horizontal offset table bit in the flags word of the font header.

End of Appendix G

METAFILE SUB-OPCODES FOR USE WITH GEM OUTPUT

The following sub-opcodes are reserved for use by the GEM Output application. GEM VDI defines sub-opcodes for the following sub-functions:

- Physical Page Size
- Coordinate Window

The opcodes are used by the GEM Output application to define how large a picture is to be rendered on the output page and also to define a transformation which maps from the metafile coordinate system to the output device.

The two GEM Output metafile sub-opcodes result in an update of the metafile header. The opcodes are not actually written to the body of the metafile.

PHYSICAL PAGE SIZE

This sub-function defines the size of the area to be output to. All of the data in the coordinate window is mapped to this area. If no physical page size is defined, the Output application will attempt a best fit on the target device, assuming that "pixels" in the metafile are square.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 3.
contrl(5)	--	Function id = 99.
contrl(6)	--	Device handle.
intin(0)	--	Sub-opcode number = 0.
intin(1)	--	Page width in tenths of millimeter.
intin(2)	--	Page height in tenths of millimeter.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

COORDINATE WINDOW

This sub-function defines the coordinate system used in the metafile. All of the data in the defined coordinate window is mapped to the area defined by the physical page size sub-function.

The coordinate window defaults to NDC space (0 to 32K). The location of the origin, (0, 0), depends on the coordinate space set when the metafile was opened (see "Open Workstation"). For example, if the Open Workstation function was invoked specifying raster coordinate space, the origin would be located in the upper left corner of the display surface.

Note that the window corner information must be specified as the lower left and upper right corners. Arbitrary opposing corners will not convey enough information.

Input

contrl(0)	--	Opcode = 5.
contrl(1)	--	Number of input vertices = 0.
contrl(3)	--	Length of intin array = 5.
contrl(5)	--	Function id = 99.
contrl(6)	--	Device handle.
intin(0)	--	Sub-opcode = 1.
intin(1)	--	x-coordinate of lower left corner of window.
intin(2)	--	y-coordinate of lower left corner of window.
intin(3)	--	x-coordinate of upper right corner of window.
intin(4)	--	y-coordinate of upper right corner of window.

Output

contrl(2)	--	Number of output vertices = 0.
contrl(4)	--	Length of intout array = 0.

METAFILE SUB-OPCODES FOR USE WITH GEM DRAW

The following sub-opcodes are reserved for use by the GEM Draw..application. GEM VDI defines the sub-opcodes for the following sub-functions:

- Start Group
- End Group
- Set Attribute Shadow On
- Set Attribute Shadow Off
- Start Draw Area Type Primitive
- End Draw Area Type Primitive
- Set No Line Style

START GROUP

This sub-function indicated the beginning of a group of primitives for the GEM Draw application. All subsequent primitives which occur before the next End Group sub-opcode will be regarded as a group by the GEM Draw application.

Input

```
contrl(0) --  Opcode = 5.
contrl(1) --  Number of input vertices = 0.
contrl(3) --  Length of intin array = 1.
contrl(5) --  Function id = 99.
contrl(6) --  Device handle.

intin(0)  --  Sub-opcode number = 10.
```

Output

```
contrl(2) --  Number of output vertices = 0.
contrl(4) --  Length of intout array = 0.
```

END GROUP	This sub-function indicated the end of a group of primitives for the GEM Draw application.
Input	contrl(0) -- Opcode = 5. contrl(1) -- Number of input vertices = 0. contrl(3) -- Length of intin array = 1. contrl(5) -- Function id = 99. contrl(6) -- Device handle. intin(0) -- Sub-opcode number = 11.
Output	contrl(2) -- Number of output vertices = 0. contrl(4) -- Length of intout array = 0.
SET NO LINE STYLE	This sub-function is used by GEM Draw to indicate that subsequent area type primitives are not to be outlined. The effects of this sub-opcode are cancelled by any subsequent set line style opcode.
Input	contrl(0) -- Opcode = 5. contrl(1) -- Number of input vertices = 0. contrl(3) -- Length of intin array = 1. contrl(5) -- Function id = 99. contrl(6) -- Device handle. intin(0) -- Sub-opcode number = 49.
Output	contrl(2) -- Number of output vertices = 0. contrl(4) -- Length of intout array = 0.

SET ATTRIBUTE SHADOW ON

This sub-function is used by GEM Draw to indicate that all subsequent primitives which occur before the next Set Attribute Shadow Off sub-opcode should be ignored because they are used to draw a drop shadow for the first primitive immediately following the Set Attribute Shadow Off sub-opcode. Internally, GEM Draw assigns a shadowed attribute to the first primitive following the Set Attribute Shadow Off sub-opcode and performs its own shadow drawing. All attribute information which occurs between Set Attribute Shadow On and Set Attribute Shadow Off will continue to be processed.

Note that GEM Draw will not drop shadows from text or from polylines consisting of only two vertices.

Input

```
contrl(0) --  Opcode = 5.
contrl(1) --  Number of input vertices = 0.
contrl(3) --  Length of intin array = 1.
contrl(5) --  Function id = 99.
contrl(6) --  Device handle.

intin(0)  --  Sub-opcode number = 50.
```

Output

```
contrl(2) --  Number of output vertices = 0.
contrl(4) --  Length of intout array = 0.
```


452

SET ATTRIBUTE SHADOW OFF

This sub-function indicates to GEM Draw the end of primitives used to draw a drop shadow of the first primitive following this sub-opcode.

Input

```
contrl(0) --   Opcode = 5.
contrl(1) --   Number of input vertices = 0.
contrl(3) --   Length of intin array = 1.
contrl(5) --   Function id = 99.
contrl(6) --   Device handle.

intin(0)  --   Sub-opcode number = 51.
```

Output

```
contrl(2) --   Number of output vertices = 0.
contrl(4) --   Length of intout array = 0.
```

START DRAW AREA TYPE PRIMITIVE

This sub-function indicates to GEM Draw that an area type primitive block follows. GEM Draw will use the vertices of the first primitive (anything except text) which follows this sub-opcode to define a GEM Draw area type primitive. All other primitives encountered before the next End Draw Area Type Primitive sub-opcode will be ignored.

Input

```
contrl(0) --   Opcode = 5.
contrl(1) --   Number of input vertices = 0.
contrl(3) --   Length of intin array = 1.
contrl(5) --   Function id = 99.
contrl(6) --   Device handle.

intin(0)  --   Sub-opcode number = 80.
```

Output

```
contrl(2) --   Number of output vertices = 0.
contrl(4) --   Length of intout array = 0.
```

END DRAW AREA TYPE PRIMITIVE	This sub-function indicates to GEM Draw the end of an area type primitive block.

Input

```
contrl(0) --  Opcode = 5.
contrl(1) --  Number of input vertices = 0.
contrl(3) --  Length of intin array = 1.
contrl(5) --  Function id = 99.
contrl(6) --  Device handle.

intin(0)  --  Sub-opcode number = 81.
```

Output

```
contrl(2) --  Number of output vertices = 0.
contrl(4) --  Length of intout array = 0.
```

End of Appendix H

INTRODUCTION

A GEM VDI bit image file is a file with extension .IMG and contains information which may be used to re-create a picture from its bit (pixel) image. The file consists of a header and raw pixel information. The pixel information may be encoded in a variety of formats.

HEADER FORMAT

The bit image file header consists of sixteen words. Each word is in machine-dependent format (for example, oriented with low byte first for Intel 808x processors or with high byte first for Motorola 68000 processors).

Word	Contents
0	upper left x of the bit image
1	upper left y of the bit image
2	lower right x of the bit image
3	lower right y of the bit image
4	source device page width
5	source device page height
6	source device pixel width in microns
7	source device pixel height in microns
8	bits per pixel
9 - 16	reserved for future use (zero)

DATA FORMAT

Pixel data may be encoded in any of four formats. All four formats may occur within the same bit image file. The four formats include:

- o run-length encoding
- o extended run-length encoding
- o raster encoding
- o raster-run encoding

The data portion of the bit image file is encoded as bytes of information.

Each of the four formats except for run-length encoding (the default) consists of a single packet prefaced by an opcode. A run-length encoded packet has no preface opcode.

Run-length Encoding

This is the default pixel data format and requires no identifying opcode (i.e., when an extended run, a raster stream, or a raster-run stream ends, run-length encoding is in effect). A run-length packet consists of two bytes of information: a run length and a pixel value. The run length must be less than 128 and the pixel value must be less than 256. A run of pixels may wrap across lines.

Run-length packet:

byte 0	run length
byte 1	pixel value

Extended Run-length Encoding

When a run is longer than 127 pixels, the extended run-length encoding may be used. An extended run includes a count of 128 pixel runs. For example, if a run exists which is 1000 pixels in length, it would be most efficient to encode it as an extended run of length seven (896 pixels) followed by a standard run of length 104.

Extended run-length packet:

byte 0	opcode = -1
byte 1	extended run length
byte 2	pixel value

Raster Encoding

Data which is not efficiently encoded in any of the other formats may be raster-encoded. A raster stream is built which consists of packed pixel values. Pixel values are packed into bytes such that each pixel value occupies the number of bits indicated in the "bits per pixel" parameter of the file header and such that a pixel value packed into a higher order portion of the byte occurs before the adjacent lower order pixel value. Pixel values should be packed across byte boundaries when necessary.

For example, assume that five pixels are to be encoded in raster format and that their pixel values are, respectively, one, two, three, four, and five. Assume that there are three bits per pixel. The resulting raster stream would be as follows:

```
pixels: 1      2      3      4      5
bits: 0 0 1  0 1 0  0 1|1  1 0 0  1 0 1   x
      |                |                  |
      |     byte 0     |      byte 1      |
```

The bit designated "x" may be either 0 or 1.

Raster packet:

byte 0	opcode = -2
byte 1	number of pixels in the stream
byte 2	packed pixel values

Raster-run Encoding

Raster-run encoding may be used to run-length encode groups of raster streams. For example, if a pattern of pixels repeats a number of times, the pattern may be packed into a raster stream (see "Raster Encoding" above) and a repeat count may be indicated.

Raster-run packet:

byte 0	opcode = -3
byte 1	pattern repeat count
byte 2	number of pixels in the stream
byte 3	packed pixel values

End of Appendix I

455

ASSIGN.SYS	Text file created by the driver installation program. Associates device identification (id) numbers with specific device driver files so that devices can be referred to by type within the application program. The ASSIGN.SYS file can be modified using any text editor.
coordinate scaling	Converting points from one space or coordinate system to another. In GEM VDI, this term refers to the change between Normalized Device Coordinates (NDC) and Raster Coordinates (RC).
coordinate systems	Cartesian space in which points are defined. GEM VDI supports two systems: Normalized Device Coordinates (NDC) and Raster Coordinates (RC).
default device driver	First driver named in the ASSIGN.SYS file. It must be the largest driver that will be loaded during a graphics session.
device driver	Device-dependent portion of GEM VDI that translates standard device-independent graphics operations to device-specific command sequences for a particular device.
device handle	Unique value used to identify which workstation the GEM VDI function should use. GEM VDI assigns these numbers at Open Workstation.
device identification number	Id number assigned to a device in the ASSIGN.SYS file. Each device in the ASSIGN.SYS file has a unique device number assigned to it.

456

face	Letter style, such as Times Roman. GEM VDI stores the definition of each style in a data file. When an application calls for the use of a particular text face, GEM VDI uses the definition to form the text characters on the specified graphics device.
font	Collection of characters all in one typeface, a subset of face.
function code	See **operation code.**
graphics command	Command that loads the GDOS into memory.
graphics device	Hardware that accepts graphics input (mouse or keyboard, for example) or displays graphics output (screen, printer, or plotter, for example).
Graphics Device Operating System (GDOS)	Device-independent portion of GEM VDI that services graphics requests and calls the device driver to send commands to graphics devices.
Generalized Drawing Primitive (GDP)	Display function used to address special device capabilities such as curve drawing. GEM VDI supports the following GDPs: bar, arc, pie, circle, ellipse, elliptical arc, elliptical pie, rounded rectangle, filled rounded rectangle, and justified graphics text. Not all devices support all GDPs.
Graphics Environment Manager Virtual Device Interface (GEM VDI)	Graphics extension to microcomputer operating systems. The GEM VDI makes it possible to run graphics applications on a microcomputer.
Graphical Kernel System (GKS)	International standard for the programming interface to graphics from an application program.
graphics primitives	Basic graphics operations performed by GEM VDI, for example, drawing lines, markers, and text strings.

hot spot	Area of the cursor that covers the pixel whose x,y location is returned during locator input. For example, the hot spot on a cross hair cursor is the intersection point of the two lines making up the cross.
metafile	Data file containing a picture description. The GEM VDI metafile can be sent to any device or used to exchange a picture between two applications.
Memory Form Definition Block (MFDB)	Block of memory that defines a raster area. An MFDB includes the following raster area information: o pointer to the memory address of the upper left corner of the first plane o height and width, in pixels o width, in words o number of planes o flag to indicate if format is standard or device-dependent o locations reserved for future use
Normalized Device Coordinate (NDC) space	Uniform virtual space by which a graphics application program can pass graphics information to a device. The GDOS maps NDCs to RCs. NDC space has its origin in the lower left corner.
normalized device coordinates (NDC)	Any point in NDC space.
operation codes (opcodes)	Passed to GDOS as part of a parameter list. The opcode indicates which graphics operation is requested.
pixel (pixel element)	Smallest element of a display surface that can be independently referenced.

---

raster area Rectangular blocks of either bits in memory or pixels on a physical device. Rasters are the steps between pixels.

Raster Coordinate (RC) space Actual device units. Raster coordinate space has its origin in the upper left corner. Its limits are determined by the resolution of the specific device.

Raster Coordinate (RC) Point in RC space.

raster functions Functions that operate on pixels either individually or in groups.

transformation mode Determines which coordinate system the application is using, NDC or RC. If NDC, the transformation mode determines how the GDOS maps the NDCs to the RCs with two methods: full NDC to RC space or uniform NDC to RC space.

Virtual Device Interface (VDI) Standard interface between device-dependent and device-independent code in a graphics environment. The GEM VDI makes all device drivers appear identical to the calling program.

virtual screen Block of memory that can be addressed as if it were a memory-mapped display.

End of Glossary

460

G

H

I