



GEM[™] Virtual Device Interface Reference Guide

GEM™

Virtual Device Interface

Reference Guide

Software Version: 2.0

COPYRIGHT

Copyright © 1986 Digital Research Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Digital Research Inc., 60 Garden Court, P.O. Box DRI, Monterey, California 93942.

DISCLAIMER

DIGITAL RESEARCH INC. MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. Further, Digital Research Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research Inc. to notify any person of such revision or changes.

NOTICE TO USER

This manual should not be construed as any representation or warranty with respect to the software named herein. Occasionally changes or variations exist in the software that are not reflected in the manual. Generally, if such changes or variations are known to exist and to affect the product significantly, a release note or READ.ME file accompanies the manual and distribution disk(s). In that event, be sure to read the release note or READ.ME file before using the product.

TRADEMARKS

Digital Research and its logo are registered trademarks of Digital Research Inc. GEM, Desktop, Graphics Environment Manager, the GEM logo, GEM Draw, GEM Output, CP/M-68K, and Concurrent are trademarks of Digital Research Inc. IBM is a registered trademark of International Business Machines, Corp. Intel is a registered trademark of Intel Corporation. Motorola is a registered trademark of Motorola Inc. Polaroid is a registered trademark of Polaroid Corporation.

First Edition: June 1986

Foreword

This guide is intended for microcomputer application programmers with operating system and graphics programming experience. It describes the GEM™ Virtual Device Interface (VDI) features, operation, and applications interface. The GEM VDI provides a device-independent environment in which to write graphics applications that can run under the most widely used operating systems.

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, making 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. The GEM VDI specifies the calling sequence to access device driver functions as well as the necessary calling parameters.

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

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.

This guide contains nine sections, seven appendices, and an index.

- Section 1 introduces the GEM VDI.
- Section 2 describes GEM VDI operating procedures, lists the functions that the VDI requires for each device, and details the VDI calling conventions.
- Sections 3 through 9 describe the various GEM VDI functions.
- Appendix A lists and describes the GEM VDI error messages.
- Appendix B describes the format of a metafile.

- **Appendix C describes the reserved metafile sub-opcodes.**
- **Appendix D defines the GEM VDI standard keyboard.**
- **Appendix E describes entry into the GEM VDI for specific operating systems and microprocessor-specific calling procedures.**
- **Appendix F describes the system font character sets and the font file format.**
- **Appendix G describes the format of a bit image file.**

The function reference table at the end of this guide lists the GEM VDI functions according to their opcode number and gives their respective C binding procedure names.

Note that throughout the remainder of this guide the term "VDI" is used to refer to the GEM Virtual Device Interface.

Contents

1 Overview

Graphics Device Operating System	1-2
Device Drivers	1-2
Opcodes.	1-3
Metafiles	1-3
Device Handles.	1-3
The Assignment File, ASSIGN.SYS	1-4
The VDI Entry Point	1-5
Coordinate Systems	1-6

2 Function Summary and Calling Conventions

Required Functions.	2-2
Function Parameters.	2-11
Running Applications Under the VDI.	2-14
Sample C Language Graphics Program.	2-17

3 Control Functions

V_OPNWK (1H)	3-2
V_CLSWK (2H).	3-12
V_CLRWK (3H).	3-13
V_UPDWK (4H)	3-14
V_OPNVWK (64H)	3-15
V_CLSVWK (65H).	3-17
VST_LOAD_FONTS (77H)	3-18
VST_UNLOAD_FONTS (78H).	3-19
VS_CLIP (81H).	3-20

4 Output Functions

V_PLINE (6H).	4-4
V_PMARKER (7H).	4-6
V_GTEXT (8H).	4-8
V_FILLAREA (9H)	4-10
V_CELLARRAY (AH).	4-12

Generalized Drawing Primitives	4-14
V_BAR (B-1H)	4-16
V_ARC (B-2H)	4-17
V_PIESLICE (B-3H)	4-19
V_CIRCLE (B-4H)	4-21
V_ELLIPSE (B-5H)	4-22
V_ELLARC (B-6H)	4-23
V_ELLPIE (B-7H)	4-24
V_RBOX (B-8H)	4-25
V_RFBOX (B-9H)	4-26
V_JUSTIFIED (B-AH)	4-27
V_CONTOURFILL (67H)	4-29
VR_RECFL (72H)	4-30
 5 Attribute Functions	
VSWR_MODE (20H)	5-3
VS_COLOR (EH)	5-6
VSL_TYPE (FH)	5-8
VSL_UDSTY (71H)	5-10
VSL_WIDTH (10H)	5-11
VSL_COLOR (11H)	5-12
VSL_ENDS (6CH)	5-13
VSM_TYPE (12H)	5-15
VSM_HEIGHT (13H)	5-17
VSM_COLOR (14H)	5-18
VST_HEIGHT (CH)	5-19
VST_POINT (6BH)	5-21
VST_ROTATION (DH)	5-23
VST_FONT (15H)	5-25
VST_COLOR (16H)	5-27
VST_EFFECTS (6AH)	5-28
VST_ALIGNMENT (27H)	5-31
VSF_INTERIOR (17H)	5-33
VSF_STYLE (18H)	5-35
VSF_COLOR (19H)	5-38
VSF_PERIMETER (68H)	5-39
VSF_UDPAT (70H)	5-40

6 Raster Operations

Memory Form Definition Block	6-2
Raster Area Formats	6-4
Coordinate Systems	6-6
Logic Operations	6-7
V_GET_PIXEL (69H)	6-9
VRO_CPYFM (6DH)	6-10
VR_TRNFM (6EH)	6-12
VRT_CPYFM (79H)	6-13

7 Input Functions

VSIN_MODE (21H)	7-3
VRQ_LOCATOR (1CH)	7-5
VSM_LOCATOR (1CH)	7-7
VRQ_VALUATOR (1DH)	7-9
VSM_VALUATOR (1DH)	7-11
VRQ_CHOICE (1EH)	7-13
VSM_CHOICE (1E)	7-14
VRQ_STRING (1FH)	7-16
VSM_STRING (1FH)	7-18
VSC_FORM (6FH)	7-20
VEX_TIMV (76H)	7-22
V_SHOW_C (7AH)	7-24
V_HIDE_C (7BH)	7-26
VQ_MOUSE (7CH)	7-27
VEX_BUTV (7DH)	7-28
VEX_MOTV (7EH)	7-30
VEX_CURV (7FH)	7-32
VQ_KEY_S (80H)	7-34

8 Inquire Functions

VQ_COLOR (1AH)	8-2
VQ_CELLARRAY (1BH)	8-4
VQL_ATTRIBUTES (23H)	8-6
VQM_ATTRIBUTES (24H)	8-8
VQF_ATTRIBUTES (25H)	8-10
VQT_ATTRIBUTES (26H)	8-12

VQ_EXTND (66H)	8-14
VQIN_MODE (73H)	8-18
VQT_EXTENT (74H)	8-19
VQT_WIDTH (75H)	8-22
VQT_NAME (82H)	8-24
VQT_FONT_INFO (83H)	8-26
VQT_JUSTIFIED (84H)	8-29
9 Escape Functions	
VQ_CHCELLS (5-1H)	9-3
V_EXIT_CUR (5-2H)	9-4
V_ENTER_CUR (5-3H)	9-5
V_CURUP (5-4H)	9-6
V_CURDOWN (5-5H)	9-7
V_CURRIGHT (5-6H)	9-8
V_CURLEFT (5-7H)	9-9
V_CURHOME (5-8H)	9-10
V_EEOS (5-9H)	9-11
V_EEOL (5-AH)	9-12
VS_CURADDRESS (5-BH)	9-13
V_CURTEXT (5-CH)	9-14
V_RVON (5-DH)	9-15
V_RVOFF (5-EH)	9-16
VQ_CURADDRESS (5-FH)	9-17
VQ_TABSTATUS (5-10H)	9-18
V_HARDCOPY (5-11H)	9-19
V_DSPCUR (5-12H)	9-20
V_RMCUR (5-13H)	9-21
V_FORM_ADV (5-14H)	9-22
V_OUTPUT_WINDOW (5-15H)	9-23
V_CLEAR_DISP_LIST (5-16H)	9-25
V_BIT_IMAGE (5-17H)	9-26
VQ_SCAN (5-18H)	9-29
V_ALPHA_TEXT (5-19H)	9-31
VS_PALETTE (5-3CH)	9-33
V_SOUND (5-3DH)	9-34
VS_MUTE (5-3EH)	9-35

VT_RESOLUTION (5-51H)	9-36
VT_AXIS (5-52H)	9-37
VT_ORGIN (5-53H)	9-38
VQ_TDIMENSIONS (5-54H)	9-39
VT_ALIGNMENT (5-55H)	9-40
VSP_FILM (5-5BH)	9-41
VQP_FILMNAME (5-5CH)	9-42
VSC_EXPOSE (5-5DH)	9-44
V_META_EXTENTS (5-62H)	9-45
V_WRITE_META (5-63H)	9-46
VM_FILENAME (5-64H)	9-47
A GEM VDI Error Messages	A-1
B GEM VDI Metafile Format	B-1
Standard Metafile Item Format	B-1
Nonstandard Metafile Items	B-4
Special Metafile Escapes	B-6
Inquire Functions	B-7
C Reserved Metafile Sub-opcodes	C-1
Metafile Sub-opcodes for GEM Output	C-1
Physical Page Size	C-2
Coordinate Window	C-3
Metafile Sub-opcodes for GEM Draw	C-4
Group	C-5
Set No Line Style	C-6
Set Attribute Shadow On	C-7
Set Attribute Shadow Off	C-8
Start Draw Area Type Primitive	C-9
End Draw Area Type Primitive	C-10
D Standard Keyboard	D-1
E Processor-Specific Data	E-1
8086-Specific Data	E-1
68000-Specific Data	E-3

Contents

F Character Sets and Font Files	F-1
Character Sets	F-1
Font Format	F-4
Font Data	F-4
Font Header	F-4
Character Offset	F-6
Horizontal Offset Table	F-7
G Bit Image File Format	G-1
Bit Image File Header	G-1
Bit Image File Data Format	G-2

Tables

1-1 Device Identification Numbers	1-4
2-1 Required VDI Functions	2-4
3-1 VDI Control Functions	3-1
3-2 Workstation Default Attribute Values	3-3
3-3 Default Color Table Index Values	3-4
4-1 VDI Output Functions	4-1
4-2 Output Functions and Line Attributes	4-2
4-3 Marker Attributes Used by v_pmarker	4-2
4-4 Output Functions and Fill Area Attributes	4-3
4-5 Output Functions and Text Attributes	4-3
5-1 VDI Attribute Functions	5-2
5-2 Writing Mode Operands	5-3
5-3 Line Type Index Values and Pattern Words	5-8
5-4 Polyline End Styles	5-13
5-5 Polymarker Types	5-15
5-6 Face Values and Names	5-25
5-7 Text Attribute Bit Mapping	5-28
6-1 Raster Operation Functions	6-1
6-2 Pixel Value to Color Index Mapping for 8-color Screens	6-5
6-3 Pixel Value to Color Index Mapping for 16-color Screens	6-6
6-4 Raster Operation Logic Operations	6-8
7-1 VDI Input Functions	7-2
7-2 Sample Mode Status Returned	7-8
8-1 VDI Inquire Functions	8-1

8-2	Face Names and Styles	8-24
9-1	Escape Function Identifiers	9-1
B-1	Control, Integer and Vertex Parameters	B-1
D-1	GEM VDI Standard Keyboard Assignments	D-1
F-1	Font Header Format	F-5
G-1	Bit Image File Header Format	G-1

Figures

1-1	Normalized Device Versus Raster Coordinates	1-7
2-1	Parameter Block Format	2-12
2-2	Output from the Polyline Sample Program	2-17
4-1	GDP Angle Specification	4-15
5-1	Points for Polyline End Styles	5-13
5-2	Character Size Definition	5-19
5-3	Angle Specification	5-23
5-4	Graphic Text Special Effects	5-29
5-5	Graphic Text Alignment	5-31
5-6	Fill Styles and Indices	5-36
6-1	Memory Form Definition Block	6-3
6-2	Standard Forms	6-5
6-3	Sample Single Plane Memory Form	6-7
8-1	Inquire Text Extent Function	8-19
8-2	Character Cell Definition	8-22
8-3	Skewed Character Offsets	8-26
F-1	GEM VDI USASCII Character Set	F-2
F-2	GEM VDI International Character Set Extension	F-3

Listings

1-1	Sample ASSIGN.SYS	1-5
2-1	Sample C Language Program, Polyline	2-18

Overview

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

The GEM VDI consists of two components: the Graphics Device Operating System (GDOS) and the device drivers and their associated font files.

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

Graphics Device Operating System

The GDOS contains the basic host and device-independent graphics functions that can be called by your application program. 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 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.

Device Drivers

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

The GEM VDI package contains drivers for the most popular microcomputer graphics devices, including screens, printers, plotters, and special cameras.

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.

A device driver produces 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.

See "Required Functions" in Section 2 for a list of the functions that the VDI requires each driver to support.

Opcodes

Each VDI function is associated with an operation code (opcode). A calling routine indicates the graphics function to be performed by a device driver by specifying the function's opcode.

References to VDI functions in this document include the function's opcode in hexadecimal notation; for example, "v_opnwk (1H)" refers to the Open Workstation function, whose opcode is one.

Metafiles

A metafile is the stored generic form of a picture file. Any VDI 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 compatible applications.

When the VDI creates a metafile, it provides the ideal device (one with square pixels). Although their origins are different, Normalized Device Coordinates (NDC) and Raster Coordinates (RC) have the same range (0 to 32767). No coordinate transformation is applied. "Coordinate Systems," later in this section, offers more information on the coordinate spaces.

Refer to Appendix B for information about the metafile format.

Device Handles

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

The GDOS returns the device handle when an application calls v_opnwk (1H), the Open Workstation function, or v_opnvwk (64H), the Open Virtual Workstation function. Routines that make subsequent calls to select a previously opened device must supply the device handle as an input argument.

The Assignment File, ASSIGN.SYS

The ASSIGN.SYS file lists the device driver filenames, their associated font filenames, device identification numbers, and any device-specific information. ASSIGN.SYS is a text file that can be created or edited using any text editor. Device ID numbers are assigned according to their type; Table 1-1 lists the device ID numbers.

The GDOS parses ASSIGN.SYS to create the assignment table, which it stores in memory to reference whenever an open workstation function is called.

You can include the R command in ASSIGN.SYS to make a driver memory resident. To do so, place "R" after the device identification number of the driver you want resident. See the example that follows Listing 1-1. The opening comments in Listing 2-1 also include an example.

Table 1-1. Device Identification Numbers

Device Type	Device ID Number
Screen	01-10
Plotter	11-20
Printer	21-30
Metafile	31-40
Camera	41-50
Tablet	51-60

Device Driver Filename

A device driver filename contained in ASSIGN.SYS must follow specific naming conventions:

- It must have eight or fewer characters
- Its first character must be alphabetic
- The file's extension must be SYS

Sample ASSIGN.SYS

Listing 1-1 shows the format of ASSIGN.SYS.

Listing 1-1. Sample ASSIGN.SYS

```
21 printer.sys
;comments, if desired
font1.fnt ;font1 description
font2.fnt ;font2 description
font3.fnt ;font3 description
01 screen.sys
;comments, if desired
font4.fnt ;font4 description
font5.fnt ;font5 description
11 plotter.sys
;comments, if desired
font6.fnt ;font6 description
font7.fnt ;font7 description
```

if included in the ASSIGN.SYS file, the following command would make the screen driver memory resident:

```
01R screen.sys
```

The VDI Entry Point

The VDI specifies the calling sequence and required parameters for access to the device driver functions.

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

- the control array
- the array of integer input parameters
- the array of input point coordinates
- the array of integer output parameters
- the array of output point coordinates

All array elements are WORD integers (two bytes). All arrays are zero-based. The content of the input and output parameter arrays is function dependent, and is included in the description of each function. "Function Parameters" in Section 2 contains more complete information about the format of the VDI function parameters.

See Appendix E for more detailed information about the microprocessor- and operating system-specific entry points into the VDI.

Coordinate Systems

All computer graphics are displayed using a coordinate system. The VDI verifies that the coordinate system of one device matches the coordinate system of another.

You can address the display surface using one of two coordinate systems:

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

The transformation mode, which the calling routine specifies when it opens a workstation, determines which coordinate system the application is to use.

Normalized Device Coordinates -- Transformation Mode 0

In the NDC system, coordinates range from 0,0 (lower left corner) to 32767,32767 (xmax,ymax in the upper right corner). The NDC system addresses the graphics display in units that are independent of the device coordinate size. These units are then transformed to Raster Coordinates by the GDOS when an application specifies transformation mode 0.

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.

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

For example, suppose that the NDC point (32767, 32767) maps to the RC point (0, 200). Because pixels are addressed at their lower left corner, the NDC point (32766, 32766) would map to the RC point (1, 199). You 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 -- Transformation Mode 2

Raster Coordinates (RC) are actual device units (for example, pixels for screens and steps for plotters and printers). Unlike the NDC system, RC system coordinates begin in the upper left corner; the lower right corner (x_{max}, y_{max}) addresses the bottom right pixel of the space. Figure 1-1 shows a comparison of the NDC and RC systems.

The GDOS does not transform coordinates when the RC system is in effect. The application needs to adjust its coordinate transformation based on the aspect ratio of the device.

The RC system eliminates the overhead involved when the GDOS transforms every point.

See the Introduction to GEM Programming for more information on coordinate system transformations.

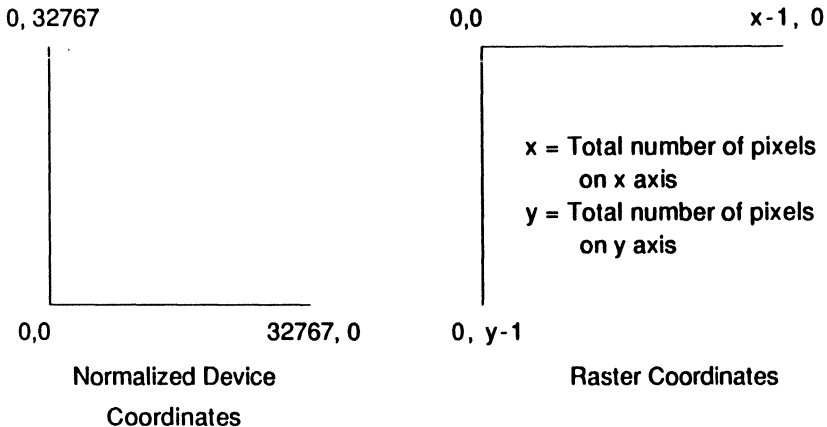


Figure 1-1. Normalized Device Versus Raster Coordinates

End of Section 1

Function Summary and Calling Conventions

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

The VDI allows you to write graphics applications in assembly language or a high-level language (C language bindings are provided). Assembly language routines address VDI functions by opcode numbers. C language routines address VDI functions by procedure name.

The VDI functions, described in sections 3 through 9, are grouped into the following categories:

- | | |
|------------------|---|
| Control | These functions open and close the graphics workstation, set default values for device attributes, load and unload fonts, clear and update the display surface, and enable or disable rectangle clipping. |
| Output | The output functions write text strings and draw output items such as lines between two or more points, a marker at one or more points, and filled areas. The output functions also provide graphics drawing primitives to produce bars, arcs, circles, ellipses and elliptical segments, rounded rectangles, and justified text. |
| Attribute | The attribute functions control the writing mode (how items are imposed over existing pixel values) and determine the characteristics of the output functions. The output function characteristics determined by the attribute functions include color, line type, marker type, fill pattern and style, and character height. |

Raster Operation	These functions operate on bit blocks in memory and on pixel blocks on physical devices. Raster operation functions copy blocks, determine the copy mode (how blocks are imposed over existing pixel values), transform block formats, and set color values for specified pixels.
Input	The input functions set the input mode to either request (wait for an input event) or sample (return input device status or location) and control the input devices. Input functions also set, show, or hide the cursor; sample the mouse button and keyboard states; and exchange the timer interrupt, button change, mouse movement, and cursor change vectors.
Inquire	The inquire functions return the current attribute settings for the output items, information about the text faces, cell definitions, and input mode.
Escape	These are device dependent control functions. For example, they can be used to control the cursor on an alpha screen, set the film type on a camera, and change the name of a metafile.

Required Functions

Each device type requires certain functions. Table 2-1, below, lists all of the VDI functions and indicates which devices are required to support them. Each device driver recognizes all opcodes, whether or not the driver supports the function. If an opcode is out of range, the driver performs no action.

Because metafiles are transportable to any device, they are required to support all those functions common to any device you may use. See Appendix B for descriptions of the metafile format, metafile support of the inquiry functions, and metafile sub-opcodes.

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

- Check the information about available features returned from Open Workstation, v_opnwk (1H), or Extended Inquire, vq_exntd (66H).
- Check the values that a function returns for the length of the integer output and output point coordinate arrays. If zero is returned where non-zero values are indicated, the driver does not support the function.
- Check the values returned from a function against the values specified in the call. If the two values are not the same, then either the driver does not support the function or the requested value is not available and the VDI selected a best fit value.

Table 2-1. Required VDI Functions

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Open Workstation, v_opnwk (1H)	X	X	X	X	X
Close Workstation, v_clswk (2H)	X	X	X	X	X
Clear Workstation, v_clrwk (3H)	X	X	X	X	X
Update Workstation, v_updwk (4H)	X	X	X	X	X
Inquire Addressable Character Cells, vq_chcells (5-1H)	X	X	X	X	X
Exit Alpha Mode, v_exitcur (5-2H)	X				
Enter Alpha Mode, v_entercur (5-3H)	X				
Cursor Up, v_curup (5-4H)	X				
Cursor Down, v_curdown (5-5H)	X				
Cursor Right, v_currright (5-6H)	X				
Cursor Left, v_curleft (5-7H)	X				
Home Cursor, v_curhome (5-8H)	X				
Erase to End of Screen, v_eeos (5-9H)	X				
Erase to End of Line, v_eeol (5-AH)	X				
Direct Cursor Address, v_curaddress (5-BH)	X				
Output Cursor Addressable Text, v_curtext (5-CH)	X				
Reverse Video On, v_rvon (5-DH)					
Reverse Video Off, v_rvoff (5-EH)					
Inquire Current Alpha Cursor Address, vq_curaddress (5-FH)	X				

Table 2-1. Required VDI Functions (Cont'd)

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Inquire Tablet Status, vq_tabstatus (5-10H)					
Hard Copy, v_hardcopy (5-11H)					
Place Graphic Cursor, v_dspcur (5-12H)	X				
Remove Last Graphic Cursor, v_rmcur (5-13H)	X				
Form Advance, v_form_adv (5-14H)		X		X	
Output Window, v_output_window (5-15H)		X		X	
Clear Display List, v_clear_disp_list (5-16H)		X		X	
Output Bit Image File, v_bit_image (5-17H)		X		X	X
Inquire Printer Scan Heights, vq_scan (5-18H)		X			
Output Printer Alpha Text, v_alpha_text (5-19H)		X		X	
Select Palette, vs_palette (5-3CH)					
Generate Tone, v_sound (5-3DH)	X				
Set/clear Muting Flag, vs_mute (5-3EH)	X				
Select Camera Film Type and Exposure, vsp_film (5-5BH)					X
Inquire Camera Film Name, vqp_filmname (5-5CH)					X
Disable/Enable Film Exposure, vsc_expose (5-5DH)					

Table 2-1. Required VDI Functions (Cont'd)

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Update Metafile Extents, v_meta_extents (5-62H)				X	
Write Metafile Item, v_write_meta (5-63H)				X	
Change GEM VDI Filename, vm_filename (5-64H)				X	
Polyline, v_line (6H)	X	X	X	X	X
Polymarker, v_pmarker (7H)	X	X	X	X	
Text, v_gtext (8H)	X	X	X	X	
Filled Area, v_fillarea (9H)	X	X	X	X	
Cell Array, v_cellarray (AH)					
Bar GDP, v_bar (B-1H)	X	X	X	X	X
Arc GDP, v_arc (B-2H)	X	X	X	X	X
Pie Slice GDP, v_pieslice (B-3H)	X	X	X	X	X
Circle GDP, v_circle (B-4H)	X	X	X	X	X
Ellipse GDP, v_ellipse (B-5H)	X	X	X	X	X
Elliptical Arc GDP, v_ellarc (B-6H)	X	X	X	X	X
Elliptical Pie Slice GDP, v_ellpie (B-7H)	X	X	X	X	X
Rounded Rectangle GDP, v_rbox (B-8H)	X	X	X	X	X
Filled, Rounded Rectangle GDP, v_rfbox (B-9H)	X	X	X	X	X
Justified Graphics Text GDP, v_justified (B-AH)	X	X	X	X	X

Table 2-1. Required VDI Functions (Cont'd)

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Set Character Height Absolute Mode, vst_height (CH)	X	X	X	X	X
Set Character Baseline Vector, vst_rotation (DH)	X	X	X	X	X
Set Color Representation, vs_color (EH)	X			X	X
Set Polyline Line Type, vsl_type (FH)	X	X	X	X	X
Set Polyline Width, vsl_width (10H)	X	X	X	X	X
Set Polyline Color Index, vsl_color (11H)	X	X	X	X	X
Set Polymarker Type, vsm_type (12H)	X	X	X	X	X
Set Polymarker Height, vsm_height (13H)	X	X	X	X	X
Set Polymarker Color Index, vsm_color (14H)	X	X	X	X	X
Set Text Font, vst_font (15H)	X	X	X	X	X
Set Text Color Index, vst_color (16H)	X	X	X	X	X
Set Fill Interior Style, vsf_interior (17H)	X	X	X	X	X
Set Fill Style Index, vsf_style (18H)	X	X	X	X	X
Set Fill Color Index, vsf_color (19H)	X	X	X	X	X
Inquire Color Representation, vq_color (1AH)	X			X	X
Inquire Cell Array, vq_cellarray (1BH)					
Input Locator - Request Mode, vrq_locator (1CH)					
Input Locator - Sample Mode, vsm_locator (1CH)					

Table 2-1. Required VDI Functions (Cont'd)

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Input Valuator - Request Mode, vrq_valuator (1DH)					
Input Valuator - Sample Mode, vsm_valuator (1DH)					
Input Choice - Request Mode, vrq_choice (1EH)					
Input Choice - Sample Mode, vsm_choice (1EH)					
Input String - Request Mode, vrq_string (1FH)	X				
Input String - Sample Mode, vsm_string (1FH)	X				
Set Writing Mode, vswr_mode (20H)	X	X		X	X
Set Input Mode, vsin_mode (21H)	X				
Inquire Current Polyline Attributes, vql_attributes (23H)	X	X	X	X	X
Inquire Current Polymarker Attributes, vqm_attributes (24H)	X	X	X	X	X
Inquire Current Fill Area Attributes, vqf_attributes (25H)	X	X	X	X	X
Inquire Current Graphic Text Attributes, vqt_attributes (26H)	X	X	X	X	X
Set Graphic Text Alignment, vst_alignment (27H)	X	X	X	X	X
Open Virtual Workstation, v_opnvwk (64H)	X				
Close Virtual Workstation, v_clsvwk (65H)	X				

Table 2-1. Required VDI Functions (Cont'd)

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Extended Inquire, vq_extnd (66H)	X	X	X	X	X
Contour Fill, v_contourfill (67H)				X	
Set Fill Perimeter Visibility, vsf_perimeter (68H)	X	X	X	X	X
Get Pixel, v_get_pixel (69H)					
Set Graphic Text Special Effects, vst_effects, (6AH)	X	X	X	X	X
Set Character Cell Height, Points Mode, vst_point (6BH)	X	X	X	X	X
Set Polyline End Styles, vsl_ends (6CH)	X	X	X	X	X
Copy Raster, Opaque, vro_cpyfm (6DH)	X				
Transform Form, vr_trn_fm (6EH)	X				
Set Mouse Form, vsc_form (6FH)	X				
Set User-defined Fill Pattern, vsf_udpat (70H)	X	X		X	X
Set User-defined Line Style, vsf_udsty (71H)	X	X		X	X
Fill Rectangle, vr_recfl (72H)	X			X	
Inquire Input Mode, vqin_mode (73H)	X				
Inquire Text Extent, vqt_extent (74H)	X	X	X		X
Inquire Character Cell Width, vqt_width (75H)	X	X	X		X
Exchange Timer Interrupt Vector, vex_timv (76H)	X				
Load Fonts, vst_load_fonts (77H)	X	X	X	X	X

Table 2-1. Required VDI Functions (Cont'd)

Function (Opcode)	Screen	Printer	Plotter	Metafile	Camera
Unload Fonts, vst_unload_fonts (78H)	X	X	X	X	X
Copy Raster, Transparent, vrt_cpyfm (79H)	X				
Show Cursor, v_show_c (7AH)	X				
Hide Cursor, v_hide_c (7BH)	X				
Sample mouse button state (7CH)	X				
Exchange Button Change Vector, vex_butv (7DH)	X				
Exchange Mouse Movement Vector, vex_motv (7EH)	X				
Exchange Cursor Change Vector, vex_curv (7FH)	X				
Sample Keyboard State Information, vq_key_s (80H)	X				
Set Clipping Rectangle, vs_clip (81H)	X	X	X	X	X
Inquire Font Name and Index, vqt_name (82H)	X	X	X		X
Inquire Current Font Information, vqt_font_info (83H)	X	X	X		X
Inquire Justified Graphics Text, vqt_justified (84H)	X				

Function Parameters

Calling routines pass arguments to the VDI functions in the form of input arrays. The VDI returns values from the functions in output arrays. All arrays are zero-based and all array elements are WORD integers. The following names are used to refer to the arrays in the descriptions of the VDI functions:

- control -- the control array
- intin -- the array of integer input parameters
- ptsin -- the array of input point coordinates
- intout -- the array of integer output parameters
- ptsout -- the array of output point coordinates

The format that the VDI functions use for parameters is shown below under "Input Parameters" and "Output Parameters."

VDI function calls can be made from applications written in assembly language or a high-level language. C language bindings are provided with GEM VDI and are shown in the description of each function.

Assembly language routines address VDI functions according to their opcodes and place the LONGWORD address of the five parameter arrays in a ten-word data structure called a Parameter Block (PB). Figure 2-1 shows the format of the PB.

OFFSET	Contents
0	Address of the control array (control)
4	Address of the integer input array (intin)
8	Address of the input point coordinate array (ptsin)
12	Address of the integer output array (intout)
16	Address of the output point coordinate array (ptsout)

Figure 2-1. Parameter Block Format

Prior to a VDI function call, the assembly language application:

- Loads the address of the PB and the VDI ID number in the appropriate registers
- Invokes the processor-specific interrupt or TRAP

See Appendix E, "Processor-Specific Data," for the registers and interrupts to be used for your system.

In the C language bindings, WORD declares a 16-bit integer type; BYTE declares an 8-bit integer type.

Input Parameters:

control(0)	Opcode number for the VDI function
control(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.
control(3)	Length of integer array intin
control(5)	Subfunction identification number for a Generalized Drawing Primitive (GDP) or Escape
control(6)	Device handle
control(7-n)	Function dependent information
intin	Array of integer input parameters
ptsin	Array of input point coordinates Refer to the Extended Inquire function, vq_extnd (66H), in Section 8 for information on how to determine the maximum size of the ptsin array.

Because coordinates may be converted by the GDOS, the calling routine must ensure that control(1), the number of vertices, is correctly set. If no x,y coordinates are being passed to the VDI by the calling routine, control(1) must be set to 0. In addition, control(3), the input integer count, must always be set. The calling routine must set control(3) to 0 if it is not passing integers to the VDI.

Output Parameters:

control(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.
control(4)	Length of integer array intout
control(6)	Device handle
control(7-n)	Function dependent information
intout	Array of integer output point parameters
ptsout	Array of output point coordinate data

The VDI always sets control(2), the output vertex count, and control(4), the output integer count, correctly. If it is not passing back information in ptsout and intout, the VDI sets control(2) and control(4) to 0.

Running Applications Under the VDI

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

1. Your program must conform to the VDI calling conventions to access graphics primitives. This process involves 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 the VDI and returns information to the calling program.

2. Adequate stack space must be made available for VDI operations. This space includes a buffer area for transforming points passed to the VDI and some fixed overhead space. "Memory Requirements," below, describes how to determine the required stack space.

3. When your program is executed, the required device drivers must be located in the current directory at the time the VDI was invoked (typically /GEMSYS). The ASSIGN.SYS file must contain the names of your device drivers and a device ID number for each device driver; refer to "The Assignment File, ASSIGN.SYS" in Section 1.
4. After successfully compiling or assembling and linking your program, you can run it like any other application, once the VDI is active. You can enable VDI graphics with the GEMVDI command, described below.

Enabling Graphics

Special commands let you enable graphics functions from the command level of the operating system. Each command loads the GDOS and any drivers declared resident in the ASSIGN.SYS file. ASSIGN.SYS and the driver files must be located in the current directory.

Note that any application to be invoked by a GEM command must be located in the search path.

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

GEMVDI /FILENAME

Enter the following command to load the VDI and start a GEM application:

GEMVDI FILENAME

To load the VDI and start the GEM Desktop.. application, enter:

GEMVDI

Memory Requirements

To determine the amount of stack space required to run a given graphics 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 array of input point coordinates 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.

Sample C Language Graphics Program

Figure 2-2 depicts the output of the sample C language graphics program contained in Listing 2-1, below. To load the VDI and start such a program use the

`GEMVDI /FILENAME`
command.

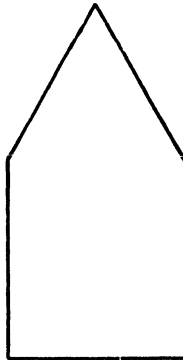


Figure 2-2. Output from the Polyline Sample Program

Listing 2-1. Sample C Language Program, Polyline

```

/* This program, sample.c, uses the VDI to draw */
/* a simple picture and then wait for a single */
/* key to be pressed before terminating.      */
/*                                             */
/* To compile sample.c:                       */
/*     lcl sample                             */
/*     lc2 sample -v                          */
/*                                             */
/* To link sample.c:                          */
/* link tcs+sample+vdibind+vdiasm,sample,    */
/*     sample/map,,                           */
/*                                             */
/* Note: Modify ASSIGN.SYS so that the screen */
/*       driver is resident (0lr XXXXXXXX.SYS) */
/*                                             */
/* To run sample.c:                            */
/*     gemvdi /sample.exe                     */
/*                                             */

/* VDI Global Arrays */
int  contrl[11];      /* control inputs      */
int  intin[80];      /* max string length  */
int  ptsin[256];     /* polygon fill points */
int  intout[45];     /* v_opnwk output     */
int  ptsout[12];     /* Points out array   */

/* Local Data Area */
int  work_in[11];    /* Input for v_opnwk  */
int  work_out[57];  /* Return from v_opnwk */
int  handle;        /* Returned handle    */
int  pxyarray[12];  /* Points for Poly Line */

int  max_length;    /* Maximum string length */
int  echo_mode;     /* Mode for char echo   */
int  echo_xy[2];    /* String x,y           */
char string[1];     /* String array         */

```

Listing 2-1. (Cont'd)

```
MAIN()                /* Called by TCS binding */
{
    int    i;          /* Loop variable          */

    /* Clear work_in array for the VDI          */
    for (i = 0; i < 11; i++)

        work_in[i] = 0;
        work_in[0] = 1;    /* Dev ID (01 = Screen) */
        work_in[1] = 1;    /* Line type (Solid)    */
        work_in[2] = 1;    /* Line Color (Black)   */
        work_in[10] = 0;   /* Use NDC coordinates  */

    v_opnwk(work_in, &handle, work_out);

    pxyarray[0] = 12000; /* v_pline vertices */
    pxyarray[1] = 12000;
    pxyarray[2] = 12000;
    pxyarray[3] = 20000;
    pxyarray[4] = 14000;
    pxyarray[5] = 24000;
    pxyarray[6] = 16000;
    pxyarray[7] = 20000;
    pxyarray[8] = 16000;
    pxyarray[9] = 12000;
    pxyarray[10] = 12000;
    pxyarray[11] = 12000;

    /* Draw the polyline                          */
    v_pline(handle, 6, pxyarray);
}
```

Listing 2-1. (Cont'd)

```
/* Length of the input string */
   max_length = sizeof(string);

   echo_mode = 0;      /* Do not echo */
   echo_xy[0] = 12000; /* x,y coordinates */
   echo_xy[1] = 12000; /* for string input */

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

   v_clswk(handle);    /* Close workstation */
}
```

End of Section 2

Control Functions

The VDI control functions initialize and close the graphics workstation, set workstation defaults, load and unload fonts, and enable or disable rectangle clipping.

Table 3-1 lists and briefly describes each of the VDI control functions.

Table 3-1. VDI Control Functions

Function	Page	Purpose
v_opnwk (1H)	3-2	Open a workstation
v_clswk (2H)	3-12	Close a workstation
v_clrwk (3H)	3-13	Clear a workstation screen
v_updwk (4H)	3-14	Execute all pending commands
v_opnvwk (64H)	3-15	Open a virtual workstation
v_clsvwk (65H)	3-17	Close a virtual workstation
vst_load_fonts (77H)	3-18	Load fonts named in ASSIGN.SYS
vst_unload_fonts (78H)	3-19	Remove device's external fonts
vs_clip (81H)	3-20	Enable/disable rectangle clipping

Workstation default attributes are listed in the description of v_opnwk (1H).

V_OPNWK (1H)

The Open Workstation function loads a graphics device driver and returns a device handle and information that describes the device's general characteristics. `v_opnwk` initializes the device's line, color, marker, text, and fill style attribute values as specified in the input arguments. See Section 5 for detailed information about attribute values. Workstation default values are listed below in Table 3-2. Table 3-3 lists the default color table index values.

If the specified device cannot be opened, `v_opnwk` returns a device handle value of zero. If the specified device is a screen, `v_opnwk` clears it and sets it to graphics mode; you do not need to call `v_clrwrk` (3H).

When `v_opnwk` is called to open a metafile, it initializes the file's buffer, writes the metafile header to the buffer, and returns the standard workstation information as described under "Output Arguments," below. See Appendix B for a description of the metafile and its header.

Use `vq_extnd` (66H) to obtain additional device-specific information.

Table 3-2. Workstation Default Attribute Values

Attribute	Default Value	VDI Function
Character height	Nominal character height	vst_height (CH)
Character baseline rotation	0 degrees rotation	vst_rotation (DH)
Text alignment	Left baseline	vst_alignment (27H)
Text effect	Normal intensity	vst_effects (6AH)
Line width	Nominal line width	vsl_width (10H)
Marker height	Nominal marker height	vsm_height (13H)
Polyline end style	Squared	vsl_ends (6CH)
Writing mode	Replace	vswr_mode (20H)
Input mode	Request for all devices	vsin_mode (21H)
Fill perimeter visibility	Visible	vsf_perimeter (68H)
User-defined line style	Solid	vsl_udsty (71H)
User-defined fill pattern	DRI Logo	vsf_udpat (70H)
Cursor	Hidden	v_show_c (7AH) and v_hide_c (7BH)
Clipping	Disabled	vs_clip (81H)

Use the VDI functions referenced in Table 3-2 to change the attribute values.

Table 3-3. Default Color Table Index Values

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

*Index 0 is defined as the current background color.

Note that devices with less than 16 colors use the first "n" index entries, where "n" is the number of available colors.

Input Arguments

- work_in[0]** Device ID number, defined as follows:
- | | |
|---------|----------|
| 01 - 10 | Screen |
| 11 - 20 | Plotter |
| 21 - 30 | Printer |
| 31 - 40 | Metafile |
| 41 - 50 | Camera |
| 51 - 60 | Tablet |
- work_in[1]** Line type value, see `vsl_type` (FH) in Section 5 for information on line types.
- work_in[2]** Polyline color index value; See Table 3-3 on page 3-4 for a list of the default color index values. `vsl_color` (11H) may also be used to set the the polyline color index (see Section 5, "Attribute Functions").
- Note:** The VDI defaults to color index 1 if the index values you specify in `work_in[2]`, `work_in[4]`, `work_in[6]`, and `work_in[9]` are out of range.
- work_in[3]** Polymarker type, see `vsm_type` (12H) in Section 5 for information on setting the polymarker type.
- work_in[4]** Polymarker color index -- see Table 3-3 on page 3-4. Use `vsm_color` (14H), described in Section 5, to set the polymarker color index.
- work_in[5]** Text font, see `vst_font` (15H) in Section 5 and `vqt_name` (82H) in Section 8 for information on face selection.
- work_in[6]** Text color index -- see Table 3-3. `vst_color` (16H), described in Section 5, should be used to set the text color index after the workstation has been opened.
- work_in[7]** Fill interior style, see `vsf_interior` (17H) and `vsf_style` (18H) in Section 5 for information on fill interior style selection.

- work_in[8]** Fill style index -- based on the fill interior style. For the pattern style, an index of 1 maps to the lowest intensity pattern on the device; if the fill interior style is solid, any index value produces an opaque fill style.
- work_in[9]** Fill color index -- see Table 3-3. vsf_color (19H), described in Section 5, should be used to set the fill color index after the workstation has been opened.
- work_in[10]** NDC to RC transformation flag:
- 0 Map full NDC space to full RC space
 - 1 Reserved
 - 2 Use RC system

Output Arguments

Note: See Table 3-2, above, for the default values of the attributes not included in the input arguments.

- handle** Device handle, used in all subsequent VDI calls to select this device
- work_out[0]** Maximum addressable x-point on the display surface in raster coordinates
- work_out[1]** Maximum addressable y-point on the display surface in raster raster coordinates
- work_out[2]** Device Coordinate units flag:
- 0 Device produces precisely scaled image
 - 1 Device does not produce a precisely scaled image
- work_out[3]** Pixel width in microns
- work_out[4]** Pixel height in microns
- work_out[5]** Number of character heights; zero indicates continuous scaling
- work_out[6]** Number of line types

- work_out[7]** Number of line widths; zero indicates continuous scaling
- work_out[8]** Number of marker types
- work_out[9]** Number of marker sizes; zero indicates continuous scaling
- work_out[10]** Number of fonts; you must load a font before sending output to devices that return zero, see **vst_load_fonts (77H)** in this section
- work_out[11]** Number of patterns
- work_out[12]** Number of hatch styles
- work_out[13]** Number of predefined colors (2 for monochrome)
- work_out[14]** Number of Generalized Drawing Primitives (GDPs)
- work_out[15]**
to
- work_out[24]** Sequential list of supported GDPs according to primitive id numbers (-1 = end of list):
 - 1 Bar -- **v_bar (B-1H)**
 - 2 Arc -- **v_arc (B-2H)**
 - 3 Pie slice -- **v_pieslice (B-3H)**
 - 4 Circle -- **v_circle (B-4H)**
 - 5 Ellipse -- **v_ellipse (B-5H)**
 - 6 Elliptical arc -- **v_ellarc (B-6H)**
 - 7 Elliptical pie -- **v_ellpie (B-7H)**
 - 8 Rounded rectangle -- **v_rbox (B-8H)**
 - 9 Filled rounded rectangle -- **v_rfbox (B-9H)**
 - 10 Justified graphics text -- **v_justified (B-AH)**

work_out[25]

to

work_out[34] Sequential list of GDP attributes:

- 0 Polyline
- 1 Polymarker
- 2 Text
- 3 Fill area
- 4 None

work_out[35] Set Color Representation, vs_color (EH), flag:

- 0 Unavailable
- 1 Available

work_out[36] Text Rotation, vst_rotation (DH), flag:

- 0 Unavailable
- 1 Available

work_out[37] Filled Area, v_fillarea (9H) flag:

- 0 Unavailable
- 1 Available

work_out[38] Cell Array, v_cellarray (AH), flag:

- 0 Unavailable
- 1 Available

work_out[39] Number of available colors; zero indicates continuous color scaling, two indicates monochrome

work_out[40] Number of available locator devices:

- 0 None
- 1 Keyboard only
- 2 Keyboard and another locator

work_out[41] Number of available valuator devices

work_out[42] Number of available choice devices

work_out[43] Number of available string devices:

- 0 None
- 1 Keyboard

work_out[44] Workstation device type:

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

work_out[45] Minimum character width in x-axis NDC/RC coordinates

work_out[46] Minimum character height in y-axis NDC/RC coordinates

work_out[47] Maximum character width in x-axis NDC/RC coordinates

work_out[48] Maximum character height in y-axis coordinates

Note that minimum and maximum character height values do not include the interline and intercharacter spacing.

work_out[49] Minimum line width in x-axis NDC/RC coordinates

work_out[50] 0

work_out[51] Maximum line width in x-axis NDC/RC coordinates

work_out[52] 0

work_out[53] Minimum marker width in x-axis NDC/RC coordinates

work_out[54] Minimum marker height in device's y-axis NDC/RC coordinates

work_out[55] Maximum marker width in x-axis NDC/RC coordinates

work_out[56] Maximum marker height in y-axis NDC/RC coordinates

Sample Call to C Language Binding

```

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

v_opnwk(work_in, &handle, work_out);

```

Parameter Block Binding

Control	Input	Output
control(0) = 1	intin(0) = work_in[0]	intout(0) = work_out[0]
control(1) = 0	intin(1) = work_in[1]	intout(1) = work_out[1]
control(2) = 6	intin(2) = work_in[2]	intout(2) = work_out[2]
control(3) = 11	intin(3) = work_in[3]	intout(3) = work_out[3]
control(4) = 45	intin(4) = work_in[4]	intout(4) = work_out[4]
control(5) = 0	intin(5) = work_in[5]	intout(5) = work_out[5]
control(6) = handle	intin(6) = work_in[6]	intout(6) = work_out[6]
	intin(7) = work_in[7]	intout(7) = work_out[7]
	.	.
	intin(10) = work_in[10]	intout(10) = work_out[10]
	.	.
		intout(44) = work_out[44]
		ptsout(0) = work_out[45]
		.
		ptsout(11) = work_out[56]

V_CLSWK (2H)

This function closes the workstation previously opened with `v_opnwk` (1H). All access to the device whose handle is specified in `v_clswk` is denied following the call. `v_clswk` performs the following close events for the indicated device type:

Screen	Selects the alpha device
Plotter	Performs an update
Printer	Performs an update
Metafile	Writes an end-of-file marker to the buffer, flushes it, and closes the file
Camera	Performs an update

Note: Use `v_clsvwk` (65H) to close all virtual workstations before calling `v_clswk`.

Input Arguments

`handle` Device handle returned from `v_opnwk` (1H)

Sample Call to C Language Binding

```
WORD v_clswk();
```

```
WORD handle;
```

```
v_clswk(handle);
```

Parameter Block Binding

Control	Input	Output
---------	-------	--------

```
control(0) = 2
control(1) = 0
control(2) = 0
control(3) = 0
control(4) = 0
control(5) = 0
control(6) = handle
```

V_CLRWK (3H)

This function clears the workstation by performing the following events for each device type:

Screen	v_clrwk clears the screen and sets it to the background color. The background color is defined as index zero in the color table (see Section 5).
Plotter	v_clrwk clears the buffer.
Printer	v_clrwk clears the buffer and issues a form feed. Use v_form_adv (5-14H) to issue a form feed without clearing the buffer (see Section 9).
Metafile	v_clrwk writes a metafile item to the metafile buffer.
Camera	v_clrwk clears the device and sets it to the currently selected background color.

Note: You do not need to call v_clrwk after calling v_opnwk (1H). v_opnwk clears the device after opening the workstation.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD      v_clrwk();
WORD      handle;
```

```
v_clrwk(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 3		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		

V_UPDWK (4H)

The Update Workstation function immediately executes the graphics commands currently stored in the specified device's buffer.

Note: You must use `v_updwk` to initiate printer output; however, `v_updwk` does not perform a form feed command. Use `v_clrwk` (3H) or `v_form_adv` (5-14H) to perform a form feed. `v_form_adv` is described in Section 9, "Escape Functions."

If the specified device is a metafile, `v_updwk` writes a metafile item to the metafile buffer.

Because they execute graphics commands upon request, `v_updwk` does not affect screens.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD v_updwk();
WORD handle;
```

```
v_updwk(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 4		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		

V_OPNVWK (64H)

Open Virtual Workstation opens and initializes multiple virtual screens on a single physical workstation. The physical workstation, whose device handle is specified in the `v_opnvwk` control array, must be a screen device previously opened with `v_opnwk` (1H). `v_opnvwk` returns a device handle for the virtual workstation in the control array. If the virtual workstation cannot be opened, `v_opnvwk` returns a device handle value of zero.

`v_opnvwk` gives each virtual workstation access to to the entire screen. The VDI maintains each workstation's attributes separately. For example, each virtual workstation may have different transformation modes and clipping rectangles.

`v_opnvwk` input and output arguments are the same as those for `v_opnwk` (1H); `v_opnwk` is described on page 3-2.

Note: The VDI has no mechanism to control input device contention between several virtual workstations. It is the application's responsibility to manage its use of the VDI input functions. Applications with multiple virtual workstations should not access input devices in sample mode. See Section 7, "Input Functions," for more information.

Input Arguments

<code>handle</code>	Screen device handle returned from <code>v_opnwk</code> (1H), specified in control array -- this is the root device handle
---------------------	--

See page 3-5 for the other input arguments.

Output Arguments

See page 3-6

Sample Call to C Language Binding

```

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

v_opnvwk(work_in, &handle, work_out);

```

Parameter Block Binding

Control	Input	Output
control(0) = 100	intin(0) = work_in[0]	intout(0) = work_out[0]
control(1) = 0	intin(1) = work_in[1]	intout(1) = work_out[1]
control(2) = 6	intin(2) = work_in[2]	intout(2) = work_out[2]
control(3) = 11	intin(3) = work_in[3]	intout(3) = work_out[3]
control(4) = 45	intin(4) = work_in[4]	intout(4) = work_out[4]
control(5) = 0	intin(5) = work_in[5]	intout(5) = work_out[5]
control(6) = handle	intin(6) = work_in[6]	intout(6) = work_out[6]
	intin(7) = work_in[7]	intout(7) = work_out[7]
	.	.
	.	.
	intin(10) = work_in[10]	intout(10) = work_out[10]
	.	.
	.	intout(44) = work_out[44]
	.	ptsout(0) = work_out[45]
	.	.
	.	ptsout(11) = work_out[56]

Important: Control(6) is both an input and an output argument. v_opnvwk returns the device handle for the virtual workstation in control(6).

V_CLSVWK (65H)

This function closes a virtual workstation previously opened with `v_opnvwk` (64H). The virtual workstation handle specified in the `v_clsvwk` call is recycled.

Input Arguments

`handle` Device handle returned from `v_opnvwk` (64H)

Sample Call to C Language Binding

```
WORD v_clsvwk();
```

```
WORD handle;
```

```
v_clsvwk(handle);
```

Parameter Block Binding

<u>Control</u>	<u>Input</u>	<u>Output</u>
<code>control(0) = 101</code>		
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 0</code>		
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VST_LOAD_FONTS (77H)

The ASSIGN.SYS file associates external fonts with a particular device driver. This function loads the driver's external fonts into memory, making them available to the calling program, and returns the number of additional font identifiers.

If the specified device's external fonts were previously loaded, or ASSIGN.SYS does not associate any font files with the driver, `vst_load_fonts` returns zero.

Note: You do not need to call `vst_load_fonts` if the driver's default system fonts are sufficient.

Input Arguments

<code>handle</code>	Device handle
<code>select</code>	Reserved, must be zero

Output Arguments

<code>additional</code>	Number of additional font identifiers
-------------------------	---------------------------------------

Sample Call to C Language Binding

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

additional = vst_load_fonts(handle, select);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 119</code>	<code>intin(0) = select</code>	<code>intout(0) = additional</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VST_UNLOAD_FONTS (78H)

This function dissociates the external fonts previously loaded by `vst_load_fonts` from the specified device. If possible, `vst_unload_fonts` frees up the memory occupied by the driver's external fonts.

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

- all workstations that share the fonts are closed, see `v_clsvwk` (65H)
- this call has been made for all workstations that share the font

The driver's default system fonts remain loaded and available.

Input Arguments

<code>handle</code>	Device handle
<code>select</code>	Reserved for future use

Sample Call to C Language Binding

```
WORD vst_unload_fonts();
WORD handle, select;
```

```
v_unload_fonts(handle, select);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 120</code>	<code>intin(0) = select</code>	
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VS_CLIP (81H)

`vs_clip` enables or disables the clipping of all output primitives by the VDI. Clipping refers to confining output to a rectangle defined within the device's display surface. Input arguments `pxarray[0]` to `pxarray[3]` specify the clipping rectangle's coordinates according to the current coordinate system (NDC or RC units). `vq_extnd` (66H), described in Section 8, indicates if clipping is currently enabled on the device (`work_out[19]`) and returns the coordinates of clipping rectangle in `work_out[45]` to `work_out[48]`.

By default, clipping is disabled when the workstation is opened.

If the specified device is a metafile, `v_clip` writes a metafile item to the metafile buffer.

Input Arguments

<code>handle</code>	Device handle
<code>clip_flag</code>	Clipping flag:
	0 Disable clipping
	1 Enable clipping
<code>pxyarray[0]</code>	x-coordinate of corner of clipping rectangle
<code>pxyarray[1]</code>	y-coordinate of corner of clipping rectangle
<code>pxyarray[2]</code>	x-coordinate of corner diagonal to <code>pxarray[0]</code>
<code>pxyarray[3]</code>	y-coordinate of corner diagonal to <code>pxarray[1]</code>

Sample Call to C Language Binding

```
WORD vs_clip();
WORD handle, clip_flag, pxyarray[4];
```

```
vs_clip(handle, clip_flag, pxyarray);
```

Parameter Block Binding

Control	Input	Output
control(0) = 129	intin(0) = clip_flag	
control(1) = 2		
control(2) = 0	ptsin(0) = pxarray[0]	
control(3) = 1	ptsin(1) = pxarray[1]	
control(4) = 0	ptsin(2) = pxarray[2]	
control(5) = 0	ptsin(3) = pxarray[3]	
control(6) = handle		

End of Section 3

Output Functions

The VDI output functions produce graphics primitives such as polylines, circles, and rounded rectangles.

Table 4-1 lists and briefly describes the output functions. Output functions use the current line, marker, fill area, or text attributes. These attributes, the output functions that use them, and the attribute functions that control them are listed in Tables 4-2 through 4-5.

Table 4-1. VDI Output Functions

Function	Page	Purpose
v_pline (6H)	4-4	Draw a line between specified points
v_pmarker (7H)	4-6	Draw markers at specified points
v_gtext (8H)	4-8	Write a text string
v_fillarea (9H)	4-10	Fill a polygon specified by points
v_cellarray (AH)*	4-12	Draw a cell array
v_bar (B-1H)	4-16	Draw a filled rectangle
v_arc (B-2H)	4-17	Draw a circular arc
v_pieslice (B-3H)	4-19	Draw a filled circular arc
v_circle (B-4H)	4-21	Draw a filled circle
v_ellipse (B-5H)	4-22	Draw a filled ellipse
v_ellarc (B-6H)	4-23	Draw an elliptical arc
v_ellpie (B-7H)	4-24	Draw a filled elliptical arc
v_rbox (B-8H)	4-25	Draw a rounded rectangle
v_rfbox (B-9H)	4-26	Draw a filled, rounded rectangle
v_justified (B-AH)	4-27	Display justified text
v_contourfill (67H)*	4-29	Seed fill a specified area
vr_recfl (72H)*	4-30	Fill a specified rectangle without perimeter

*These functions are either not required and may not be available on all devices, or contain unrequired options; see "Required Functions" in Section 2.

Table 4-2. Output Functions and Line Attributes

Output Functions That Use Line Attributes			
v_pline (6H)	v_arc (B-2H)	v_ellarc (B-6H)	v_rbox (B-8H)
<u>Line Attributes:</u>		<u>Attribute Functions:</u>	
Line type		vsl_type (FH)	
Line width		vsl_width (10H)	
Line color		vsl_color (11H)	
Line end style		vsl_ends (6CH)	

Table 4-3. Marker Attributes Used by v_pmarker

Marker Attributes	Attribute Functions
Marker type	vsm_type (12H)
Marker height	vsm_height (13H)
Marker color	vsm_color (14H)

Note: The attribute functions are described in Section 5.

Table 4-4. Output Functions and Fill Area Attributes

Output Functions That Use Fill Area Attributes		
v_fillarea (9H) v_circle (B-4H) v_rfbbox (B-9H)	v_bar (B-1H) v_ellipse (B-5H) v_contourfill (67H)	v_pieslice (B-3H) v_ellpie (B-7H) vr_recfl (72H)*
<u>Fill Area Attributes:</u> Fill interior style Fill style index Fill area color Fill perimeter	<u>Attribute Functions:</u> vsf_interior (17H) vsf_style (18H) vsf_color (19H) vsf_perimeter (68H)	

*vr_recfl always operates with the fill perimeter disabled.

Table 4-5. Output Functions and Text Attributes

Output Functions That Use Text Attributes	
v_gtext (8H)	v_justified (B-AH)
<u>Text Attributes:</u> Character height Baseline vector Text face Text color Text alignment Text effect Cell height, points	<u>Attribute Functions:</u> vst_height (CH) vst_rotation (DH) vst_font (15H) vst_color (16H) vst_alignment (27H) vst_effects (6AH) vst_point (6BH)

Note: The output functions operate according to the current writing mode as controlled by vswr_mode (20H). vswr_mode is described in Section 5. vs_clip (81H), described in Section 3, defines the area on the screen where an output function is allowed to draw.

V_PLINE (6H)

This function displays a polyline on the graphics device. The line's starting point is specified in the first coordinate pair ($xy[0]$ and $xy[1]$). `v_pline` draws the line between subsequent points as specified in the remaining input arguments according to the current coordinate system (NDC or RC units).

`v_pline` displays a zero length line (degenerate case) as a point and does not display a single coordinate pair. Use `vq_extnd` (66H), described in Section 8, "Inquire Functions," to obtain the maximum number of input vertices for the specified device.

`v_pline` produces lines according to the line attributes listed in Table 4-2, above, and the current writing mode. You can use `vql_attributes` (23H) to obtain the current settings of the line attributes.

Input Arguments

<code>handle</code>	Device handle
<code>count</code>	Number of input vertices (n)
<code>xy[0]</code>	x-coordinate of first point
<code>xy[1]</code>	y-coordinate of first point
<code>xy[2]</code>	x-coordinate of second point
<code>xy[3]</code>	y-coordinate of second point
.	
.	
<code>xy[2*count-2]</code>	x-coordinate of last point
<code>xy[2*count-1]</code>	y-coordinate of last point

Sample Call to C Language Binding

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

```
v_pline(handle, count, xy);
```

Parameter Block Binding

Control	Input
control(0) = 6	ptsin(0) = xy[0]
control(1) = count	ptsin(1) = xy[1]
control(2) = 0	ptsin(2) = xy[2]
control(3) = 0	ptsin(3) = xy[3]
control(4) = 0	.
control(5) = 0	.
control(6) = handle	ptsin(2*count-2) = xy[2*count-2] ptsin(2*count-1) = xy[2*count-1]

V_PMARKER (7H)

This function draws markers at the points specified in the input argument coordinates (in NDC or RC units). `v_pmarker` produces markers according to the attributes listed in Table 4-3 and the current writing mode. `vqm_attributes` (24H), described in Section 8, returns the current settings of these attributes.

Use `vq_extnd` (66H), to obtain the maximum number of polymarker vertices for the specified device.

Input Arguments

<code>handle</code>	Device handle
<code>count</code>	Number of input vertices (n)
<code>xy[0]</code>	x-coordinate of first marker
<code>xy[1]</code>	y-coordinate of first marker
<code>xy[2]</code>	x-coordinate of second marker
<code>xy[3]</code>	y-coordinate of second marker
.	.
<code>xy[2*count-2]</code>	x-coordinate of last marker
<code>xy[2*count-1]</code>	y-coordinate of last marker

Sample Call to C Language Binding

```
WORD v_marker();  
WORD handle, count, xy[2 * count]  
  
v_pmarker(handle, count, xy);
```

Parameter Block Binding

Control	Input
control(0) = 7	ptsin(0) = xy[0]
control(1) = count	ptsin(1) = xy[1]
control(2) = 0	ptsin(2) = xy[2]
control(3) = 0	ptsin(3) = xy[3]
control(4) = 0	.
control(5) = 0	.
control(6) = handle	ptsin(2*count-2) = xy[2*count-2]
	ptsin(2*count-1) = xy[2*count-1]

V_GTEXT (8H)

`v_gtext` writes a text string to the display surface. The (x,y) position specified by the calling routine is the alignment point of the text string in units of the current coordinate system (NDC or RC). The `vst_alignment` function establishes the relationship between the string's starting point and the specified x,y position (see in Section 5). See Table 4-5, above, for other text attributes that affect the operation of `v_gtext`.

Note that `vqt_attributes` (26H) returns the current text attribute settings (see Section 8, "Inquire Functions").

For the C language binding, the string input argument is byte-oriented and must be null-terminated. For the Parameter Block (assembly language) binding, the string is word-oriented. Any unsupported character is mapped to a question mark (?), the symbol for an undefined character. `work_out[15]` of `vq_extnd` (66H) returns the maximum number of characters (size of `intin` array) allowed in the text string. `vq_extnd` is described in Section 8.

Be sure that the specified device has a font loaded before outputting text. Printers and some other devices may not have a system font available; if `v_opnwk` (1H) returned zero in `work_out[10]` when the device was opened, call `vst_load_fonts` (77H) before `v_gtext`. `v_opnwk` and `vst_load_fonts` are described in Section 3, "Control Functions."

Input Arguments

<code>handle</code>	Device handle
<code>x</code>	x-coordinate of text string's alignment point
<code>y</code>	y-coordinate of text string's alignment point
<code>string</code>	ASCII character string

Sample Call to C Language Binding

```
WORD v_gtext();  
WORD handle, x, y;  
BYTE *string;
```

```
v_gtext(handle, x, y, string);
```

Parameter Block Binding

Control	Input	Output
control(0) = 8	intin = string (in 16-bit	
control(1) = 1	characters)	
control(2) = 0		
control(3) = number of	ptsin(0) = x\	
chars in string		
control(4) = 0	ptsin(1) = y	
control(5) = 0		
control(6) = handle		

V_FILLAREA (9H)

The `v_fillarea` function fills the polygon defined by the input argument coordinates (in NDC or RC units) according to the fill area attributes listed in Table 4-4 and the writing mode. Use `vqf_attributes` (25H) to obtain the current settings of the attributes that affect the operation of this function (see Section 8, "Inquire Functions").

When the fill perimeter attribute is enabled (visible), `v_fillarea` outlines the polygon with a solid line of the current fill area color. If the specified device does not have fill area capability, the polygon is outlined with the current fill area color.

A polygon with zero area (degenerate case) is displayed as a dot only when the fill perimeter attribute is enabled. If the fill perimeter attribute is disabled, a dot is not displayed for the degenerate case.

`v_fillarea` does not display a polygon with only one vertex. The device driver ensures that the fill area is closed by connecting the first point to the last point.

`work_out[14]` of `vq_extnd` (66H) returns the maximum number of input vertices you may use to define a polygon.

Input Arguments

<code>handle</code>	Device handle
<code>count</code>	Number of input vertices (n)
<code>xy[0]</code>	x-coordinate of first point
<code>xy[1]</code>	y-coordinate of first point
<code>xy[2]</code>	x-coordinate of second point
<code>xy[3]</code>	y-coordinate of second point
.	.
<code>xy[2*count-2]</code>	x-coordinate of last point

`xy[2*count-1]` y-coordinate of last point

Sample Call to C Language Binding

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

```
v_fillarea(handle, count, xy);
```

Parameter Block Binding

Control	Input
<code>control(0) = 9</code>	<code>ptsin(0) = xy[0]</code>
<code>control(1) = count</code>	<code>ptsin(1) = xy[1]</code>
<code>control(2) = 0</code>	<code>ptsin(2) = xy[2]</code>
<code>control(3) = 0</code>	<code>ptsin(3) = xy[3]</code>
<code>control(4) = 0</code>	.
<code>control(5) = 0</code>	.
<code>control(6) = handle</code>	<code>ptsin(2*count-2) = xy[2*count-2]</code>
	<code>ptsin(2*count-1) = xy[2*count-1]</code>

V_CELLARRAY (AH)

This function draws the rectangular array defined by the pxyarray input coordinates (in NDC or RC units) and divides it into color cells of equal size and shape based on the length and number of each row in the color index array. The color index array determines the color for each cell; it is composed of indices selected from the color table (see Section 5).

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

If the specified device does not support cell arrays, the rectangle is outlined with a solid line in the current line color and line width.

Use the vq_cellarray (1BH) function to obtain information about a previously defined cell array -- see Section 8, "Inquire Functions."

Note: v_cellarray may not be supported by all devices drivers. work_out[38] of v_opnwk (1H) returns zero if the device does not provide this function.

Input Arguments

handle	Device handle
row_length	Length of each row in color index array (size as declared in a high-level language)
el_used	Number of elements used in each row (number of columns) of color index array
num_rows	Number of rows in color index array
wrt_mode	Pixel operation (write mode) to be performed: 1 Replace 2 Transparent 3 XOR 4 Reverse transparent

See vswr_mode (20H) in Section 5 for a description of each mode.

colarray[0] Color index array, stored by row
 .
 .
 colarray[n] Last row of color index
 The size of the array equals (num_rows * el_used).
 pxyarray[0] x-coordinate of rectangle's lower left corner
 pxyarray[1] y-coordinate of rectangle's lower left corner
 pxyarray[2] x-coordinate of rectangle's upper right corner
 pxyarray[3] y-coordinate of rectangle's upper right corner

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 10	intin(0) = colarray[0]	
control(1) = 2	.	
control(2) = 0	.	
control(3) = n	intin(n) = colarray[n]	
control(4) = 0		
control(5) = 0	ptsin(0) = pxyarray[0]	
control(6) = handle	ptsin(1) = pxyarray[1]	
control(7) = row_length	ptsin(2) = pxyarray[2]	
control(8) = el_used	ptsin(3) = pxyarray[3]	
control(9) = num_rows		
control(10) = wrt_mode		

Generalized Drawing Primitives

The Generalized Drawing Primitives (GDPs) produce special graphics elements such as arcs, circles, and ellipses.

All GDPs share the same opcode; however, input and output arguments and the arrays are primitive-dependent. The calling routine specifies the function to be performed according to the primitive's ID number, which is contained in control(5). The primitive ID numbers are listed below:

- | | |
|----|--|
| 1 | Bar -- v_bar |
| 2 | Arc -- v_arc |
| 3 | Pie slice -- v_pieslice |
| 4 | Circle -- v_circle |
| 5 | Ellipse -- v_ellipse |
| 6 | Elliptical arc -- v_ellarc |
| 7 | Elliptical pie -- v_ellpie |
| 8 | Rounded rectangle -- v_rbox |
| 9 | Filled rounded rectangle -- v_rfbox |
| 10 | Justified graphics text -- v_justified |

GDP Angle Specifications

All GDP input angle specifications must be in tenths of degrees. 0 degrees is 90 degrees to the right of vertical. As shown in Figure 4-1, values increase in the counterclockwise direction. Arcs are drawn counterclockwise.

All radius arguments, except for `v_ellipse` (B-5H) and `v_ellarc` (B-6H), specify an extent (distance) in x-axis units. The `v_ellipse` and `v_ellarc` functions use both x and y axis units.

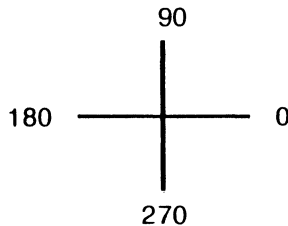


Figure 4-1. GDP Angle Specification

Note that all GDP coordinates are specified in units of the current coordinate system (NDC or RC).

Descriptions of the individual GDP functions follow.

V_BAR (B-1H)

This function draws a rectangle using the current writing mode and fill area attributes. These attributes are listed in Table 4-4, above; use the `vfq_attributes` function (25H) to obtain their current settings.

Input Arguments

`pxyarray[0]` x-coordinate of corner of bar
`pxyarray[1]` y-coordinate of corner of bar
`pxyarray[2]` x-coordinate of corner diagonal to `pxyarray[0]`
`pxyarray[3]` y-coordinate of corner diagonal to `pxyarray[1]`

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>ptsin(0) = pxyarray[0]</code>	
<code>control(1) = 2</code>	<code>ptsin(1) = pxyarray[1]</code>	
<code>control(2) = 0</code>	<code>ptsin(2) = pxyarray[2]</code>	
<code>control(3) = 0</code>	<code>ptsin(3) = pxyarray[3]</code>	
<code>control(4) = 0</code>		
<code>control(5) = 1</code>		
<code>control(6) = handle</code>		

V_ARC (B-2H)

`v_arc` draws an arc using the current writing mode and polyline attributes. These attributes are listed in Table 4-2, above; use `vql_attributes` (23H) to obtain their current settings.

Arcs are defined for `v_arc` by the beginning and ending angles, center point (x- and y-coordinates), and the length of the radius as measured on the x axis.

Input Arguments

<code>handle</code>	Device handle
<code>begang</code>	Beginning angle in tenths of degrees, (0-3600)
<code>endang</code>	Ending angle in tenths of degrees, (0-3600)
<code>x</code>	x-coordinate of arc's center point
<code>y</code>	y-coordinate of arc's center point
<code>radius</code>	Length of arc's radius in x-axis units

Sample Call to C Language Binding

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

Parameter Block Binding

Control	Input	Output
control(0) = 11	intin(0) = belang	
control(1) = 4	intin(1) = endang	
control(2) = 0		
control(3) = 2	ptsin(0) = x	
control(4) = 0	ptsin(1) = y	
control(5) = 2	ptsin(2) = 0	
control(6) = handle	ptsin(3) = 0	
	ptsin(4) = 0	
	ptsin(5) = 0	
	ptsin(6) = radius	
	ptsin(7) = 0	

V_PIESLICE (B-3H)

This function draws a pie slice using the current writing mode and fill area attributes. Table 4-4, above, lists the fill area attributes. Use the `vqf_attributes` function (25H) to obtain the current attribute settings.

Specify the pie slice by its beginning and ending angles, center point (x- and y-coordinates), and the length of its radius, measured on the x axis. `v_pieslice` draws the pieslice counter-clockwise from the beginning angle to the end angle.

Input Arguments

<code>handle</code>	Device handle
<code>begang</code>	Beginning angle in tenths of degrees, (0-3600)
<code>endang</code>	Ending angle in tenths of degrees, (0-3600)
<code>x</code>	x-coordinate of center point of pie slice
<code>y</code>	y-coordinate of center point of pie slice
<code>radius</code>	Length of radius of pie slice in x-axis units

Sample Call to C Language Binding

```
WORD v_pieslice();  
WORD handle, x, y, radius, begang, endang;  
  
v_pieslice(handle, x, y, radius, begang, endang);
```

Parameter Block Binding

Control	Input	Output
control(0) = 11	intin(0) = began	
control(1) = 4	intin(1) = endang	
control(2) = 0		
control(3) = 2	ptsin(0) = x	
control(4) = 0	ptsin(1) = y	
control(5) = 3	ptsin(2) = 0	
control(6) = handle	ptsin(3) = 0	
	ptsin(4) = 0	
	ptsin(5) = 0	
	ptsin(6) = radius	
	ptsin(7) = 0	

V_CIRCLE (B-4H)

This function draws a circle using the current writing mode and fill area attributes. These attributes are listed in Table 4-4, above; use `vqf_attributes (25H)` to obtain their current settings.

Specify the circle by the *x*- and *y*-coordinates of its center point and the length of its radius as measured on the *x*-axis.

Input Arguments

<code>handle</code>	Device handle
<code>x</code>	<i>x</i> -coordinate of circle's center point
<code>y</code>	<i>y</i> -coordinate of circle's center point
<code>radius</code>	Length circle's of radius in <i>x</i> -axis units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>ptsin(0) = x</code>	
<code>control(1) = 3</code>	<code>ptsin(1) = y</code>	
<code>control(2) = 0</code>	<code>ptsin(2) = 0</code>	
<code>control(3) = 0</code>	<code>ptsin(3) = 0</code>	
<code>control(4) = 0</code>	<code>ptsin(4) = radius</code>	
<code>control(5) = 4</code>	<code>ptsin(5) = 0</code>	
<code>control(6) = handle</code>		

V_ELLIPSE (B-5H)

This function draws an ellipse using the writing mode and fill area attributes. These attributes are listed in Table 4-4. `vqf_attributes (25H)` returns the current settings of the attributes that affect the operation of `v_ellipse`.

An ellipse is defined for `v_ellipse` by the *x*- and *y*-coordinates of the ellipse's center point, the *x*-axis radius, and the *y*-axis radius.

Input Arguments

<code>handle</code>	Device handle
<code>x</code>	<i>x</i> -coordinate of ellipse's center point
<code>y</code>	<i>y</i> -coordinate of ellipse's center point
<code>xradius</code>	Radius in <i>x</i> -axis units
<code>yradius</code>	Radius in <i>y</i> -axis units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>ptsin(0) = x</code>	
<code>control(1) = 2</code>	<code>ptsin(1) = y</code>	
<code>control(2) = 0</code>	<code>ptsin(2) = xradius</code>	
<code>control(3) = 0</code>	<code>ptsin(3) = yradius</code>	
<code>control(4) = 0</code>		
<code>control(5) = 5</code>		
<code>control(6) = handle</code>		

V_ELLARC (B-6H)

This function draws an elliptical arc. It uses the writing mode and line attributes. The line attributes are listed in Table 4-2; use `vql_attributes (23H)` to obtain their current settings.

Define an elliptical arc for `v_ellarc` by specifying its beginning and ending angles, the *x*- and *y*-coordinates of its center point, its *x*-axis radius, and its *y*-axis radius.

Input Arguments

<code>handle</code>	Device handle
<code>begang</code>	Beginning angle in tenths of degrees, (0-3600)
<code>endang</code>	Ending angle in tenths of degrees, (0-3600)
<code>x</code>	<i>x</i> -coordinate of arc's center point
<code>y</code>	<i>y</i> -coordinate of arc's center point
<code>xradius</code>	Radius in <i>x</i> -axis units
<code>yradius</code>	Radius in <i>y</i> -axis units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>intin(0) = begang</code>	
<code>control(1) = 2</code>	<code>intin(1) = endang</code>	
<code>control(2) = 0</code>		
<code>control(3) = 2</code>	<code>ptsin(0) = x</code>	
<code>control(4) = 0</code>	<code>ptsin(1) = y</code>	
<code>control(5) = 6</code>	<code>ptsin(2) = xradius</code>	
<code>control(6) = handle</code>	<code>ptsin(3) = yradius</code>	

V_ELLPIE (B-7H)

This function draws an elliptical pie segment using the writing mode and fill area attributes. Table 4-4 lists the fill attributes; use `vqf_attributes` (25H) to obtain their current settings.

Define the elliptical pie segment for `v_ellpie` by specifying its beginning and ending angles, the x- and y-coordinates of its center point, its x-axis radius, and its y-axis radius.

Input Arguments

<code>handle</code>	Device handle
<code>begang</code>	Beginning angle in tenths of degrees, (0-3600)
<code>endang</code>	Ending angle in tenths of degrees, (0-3600)
<code>x</code>	x-coordinate of arc's center point
<code>y</code>	y-coordinate of arc's center point
<code>xradius</code>	Radius in x-axis units
<code>yradius</code>	Radius in y-axis units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>intin(0) = begang</code>	
<code>control(1) = 2</code>	<code>intin(1) = endang</code>	
<code>control(2) = 0</code>		
<code>control(3) = 2</code>	<code>ptsin(0) = x</code>	
<code>control(4) = 0</code>	<code>ptsin(1) = y</code>	
<code>control(5) = 7</code>	<code>ptsin(2) = xradius</code>	
<code>control(6) = handle</code>	<code>ptsin(3) = yradius</code>	

V_RBOX (B-8H)

`v_rbox` draws a rectangle with rounded corners. This function uses the current writing mode and line attributes. See Table 4-2, above. `vgl_attributes` (23H) returns the current settings of the attributes that affect `v_rbox`.

Define the rectangle by specifying coordinates (NDC or RC units) for two of its corners.

Input Arguments

Handle	device handle
<code>xyarray[0]</code>	x-coordinate of rectangle corner
<code>xyarray[1]</code>	y-coordinate of rectangle corner
<code>xyarray[2]</code>	x-coordinate of corner diagonal to <code>xyarray[0]</code>
<code>xyarray[3]</code>	y-coordinate of corner diagonal to <code>xyarray[1]</code>

Sample Call to C Language Binding

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

v_rbox(handle, xyarray);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>ptsin(0) = xyarray[0]</code>	
<code>control(1) = 2</code>	<code>ptsin(1) = xyarray[1]</code>	
<code>control(2) = 0</code>	<code>ptsin(2) = xyarray[2]</code>	
<code>control(3) = 0</code>	<code>ptsin(3) = xyarray[3]</code>	
<code>control(4) = 0</code>		
<code>control(5) = 8</code>		
<code>control(6) = handle</code>		

V_RFBOX (B-9H)

This function draws a filled rectangle with rounded corners. `v_rfbbox` uses the current writing mode and fill area attributes. Table 4-2, above, lists the fill area attributes; use `vqf_attributes` (25H) to obtain their current settings.

Define the rounded rectangle to be filled by specifying the coordinates (NDC or RC units) for two of its corners.

Input Arguments

Handle	device handle
<code>xyarray[0]</code>	x-coordinate of rectangle corner
<code>xyarray[1]</code>	y-coordinate of rectangle corner
<code>xyarray[2]</code>	x-coordinate of corner diagonal to <code>xyarray[0]</code>
<code>xyarray[3]</code>	y-coordinate of corner diagonal to <code>xyarray[1]</code>

Sample Call to C Language Binding

```
WORD v_rfbbox();
WORD handle, xyarray[4];

v_rfbbox(handle, xyarray);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 11</code>	<code>ptsin(0) = xyarray[0]</code>	
<code>control(1) = 2</code>	<code>ptsin(1) = xyarray[1]</code>	
<code>control(2) = 0</code>	<code>ptsin(2) = xyarray[2]</code>	
<code>control(3) = 0</code>	<code>ptsin(3) = xyarray[3]</code>	
<code>control(4) = 0</code>		
<code>control(5) = 9</code>		
<code>control(6) = handle</code>		

V_JUSTIFIED (B-AH)

This function writes and attempts to justify both the left and right margins of a text string. The text string is justified according to the alignment points, passed as input arguments *x* and *y*, and the requested string length. *x*, *y*, and length are specified in units of the current coordinate system (NDC or RC).

v_justified uses the current writing mode and text alignment attributes; see Table 4-5. Use *vqt_attributes* (26H) to obtain the current settings of the attributes that affect the operation of *v_justified*.

You can specify whether or not *v_justified* is to modify the inter-word and/or inter-character spacing to match the requested length of the text string with the *word_space* and *char_space* arguments.

For the C language binding, the string input argument is byte-oriented and must be null-terminated. For the Parameter Block (assembly language) binding, the string is word-oriented. Any unsupported character is mapped to a question mark (?), the symbol for an undefined character. *vq_extnd* (66H), described in Section 8, returns the maximum number of characters (size of *intin* array) allowed in the text string.

The *vqt_justified* function (84H), also described in Section 8, returns the *x*- and *y*-axis offsets for each character in a text string given the same input arguments as *v_justified*.

Be sure that the specified device has a font loaded before outputting text. Printers and some other devices may not have a system font available; if *v_opnwk* (1H) returned zero in *work_out*[10] when the device was opened, call *vst_load_fonts* (77H) before *v_justified*. *v_opnwk* and *vst_load_fonts* are described in Section 3, "Control Functions."

Input Arguments

word_space	Word spacing flag: zero Do not modify inter-word spacing nonzero Modify inter-word spacing
char_space	Character spacing flag: zero Do not modify inter-character spacing nonzero Modify inter-character spacing
string	ASCII character string
x	x-coordinate of the text alignment point
y	y-coordinate of the text alignment point
length	Requested length of the string, in x-axis units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 11	intin(0) = word_space	
control(1) = 2	intin(1) = char_space	
control(2) = 0	intin(n+2) = string(n)	
control(3) = 2+n		
control(4) = 0	ptsin(0) = x	
control(5) = 10	ptsin(1) = y	
control(6) = handle	ptsin(2) = length	
	ptsin(3) = 0	

Note: "n" is used here to represent the number of characters in the string.

V_CONTOURFILL (67H)

This function is often referred to as “seed” or “flood” fill. It fills an area using the current fill area attributes until it reaches either the edge of the screen or a specified color. Table 4-4 lists the fill area attributes; use `vqf_attributes` (25H) to obtain their current settings.

The center point of the area to be filled is specified in the `x` and `y` arguments using units of the current coordinate system. The color that defines the contour to be filled by `v_contourfill` is specified as a fill area color index value. If this value is negative, `v_contourfill` searches for any color other than the color of the seed point.

Note: This function may not be supported by all device drivers. Check `work_out[7]` of `vq_extnd` (66H) to determine if the device has seed fill capability.

Input Arguments

<code>index</code>	Color index that defines the contour
<code>x</code>	x-coordinate of starting point
<code>y</code>	y-coordinate of starting point

Sample Call to C Language Binding

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

```
v_contourfill(handle, x, y, index)
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 103</code>	<code>intin(0) = index</code>	
<code>control(1) = 1</code>		
<code>control(2) = 0</code>	<code>ptsin(0) = x</code>	
<code>control(3) = 1</code>	<code>ptsin(1) = y</code>	
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VR_RECFL (72H)

vr_recfl fills a rectangle whose area is defined by the pxyarray arguments in units of the current coordinate system (NDC or RC) according to the fill area attributes. This function is similar to v_bar (B-1H) except that the fill perimeter attribute is not used. Regardless of its current setting, vr_recfl operates with the fill perimeter attribute disabled (invisible).

The fill area attributes are listed in Table 4-4, above; vqf_attributes (25H) returns their current settings.

Input Arguments

pxyarray[0] x-coordinate of corner of rectangle
 pxyarray[1] y-coordinate of corner of rectangle
 pxyarray[2] x-coordinate of corner diagonal to pxyarray[0]
 pxyarray[3] y-coordinate of corner diagonal to pxyarray[1]

Sample Call to C Language Binding

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

```
v_recfl(handle, pxyarray);
```

Parameter Block Binding

Control	Input	Output
control(0) = 114	ptsin(0) = pxyarray[0]	
control(1) = 2	ptsin(1) = pxyarray[1]	
control(2) = 0	ptsin(2) = pxyarray[2]	
control(3) = 0	ptsin(3) = pxyarray[3]	
control(4) = 0		
control(5) = 0		
control(6) = handle		

End of Section 4

Attribute Functions

The VDI attribute functions control how output items are imposed over existing pixel values (the writing mode) and determine the characteristics of the output functions. The output function characteristics determined by the attribute functions include color, line type, marker type, fill pattern and style, and character height.

Note that the function descriptions in this section are organized according to the attributes they control, rather than opcode.

Table 5-1 lists each of the VDI attribute functions.

Table 5-1. VDI Attribute Functions

Function	Page	Purpose
vst_height (CH)	5-19	Set character height, absolute mode
vst_rotation (DH)	5-23	Set the character baseline vector
vs_color (EH)	5-6	Determine color representation
vsl_type (FH)	5-8	Set the line type
vsl_width (10H)	5-11	Set the line width
vsl_color (11H)	5-12	Set the line color index
vsm_type (12H)	5-15	Set the polymarker type
vsm_height (13H)	5-17	Set polymarker height
vsm_color (14H)	5-18	Set the polymarker color index
vst_font (15H)	5-25	Select a character font
vst_color (16H)	5-27	Set the text color index
vsf_interior (17H)	5-33	Set the fill interior style
vsf_style (18H)	5-35	Set the fill style index
vsf_color (19H)	5-38	Set the fill color index
vswr_mode (20H)	5-3	Select the writing mode
vst_alignment (27H)	5-31	Set text alignment
vsf_perimeter (68H)	5-39	Determine fill area perimeter visibility
vst_effects (6AH)	5-28	Set special effects for text
vst_point (6BH)	5-21	Set character height, points mode
vsl_ends (6CH)	5-13	Set line end styles
vsf_udpat (70H)	5-40	Set a user-defined fill pattern
vsl_udsty (71H)	5-10	Set a user-defined line style

VSWR_MODE (20H)

This function selects the writing mode. The writing mode affects the way new pixels for subsequent drawing operations are placed on the display. The calling routine specifies the operation to be performed between the color indices of the current pixel (source) and the existing pixel (destination) by selecting one of four writing modes.

The writing modes are: replace, transparent, XOR, and reverse transparent. `work_out[9]` of `vq_extnd (66H)` returns the number of writing modes available on a particular device -- see Section 8. Table 5-2 defines the operands used in the Boolean expressions that describe the four writing modes.

Table 5-2. Writing Mode Operands

Operand	Definition
mask	Line style or fill pattern
fore	Selected color after mapping from the VDI
back	Color 0 after mapping from the VDI (white is default)
old	Current color value
new	Replacement color value

Mode 1, Replace

Replace mode is insensitive to the currently displayed image. Any information already displayed is replaced. This is the default writing mode when the workstation is opened. The Boolean expression for replace mode is:

$$\text{new} = (\text{fore AND mask}) \text{ OR } (\text{back AND NOT mask})$$

Mode 2, Transparent

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

$$\text{new} = (\text{fore AND mask}) \text{ OR } (\text{old AND NOT mask})$$

Mode 3, XOR

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

$$\text{new} = \text{mask XOR old}$$

Mode 4, Reverse Transparent

Reverse transparent mode affects only the pixels where the mask is 0, changing them to the fore value. The Boolean expression for this mode is:

$$\text{new} = (\text{old AND mask}) \text{ OR } (\text{fore AND NOT mask})$$

Input Arguments

handle	Device handle
mode	Writing mode: 1 Replace mode 2 Transparent mode 3 XOR mode 4 Reverse transparent mode

If the writing mode specified by the calling routine is not in the range of 1 - 4, vswr_mode selects replace mode, 1.

Output Arguments

`set_mode` Selected writing mode

Sample Call to C Language Binding

```
WORD vswr_mode();
WORD set_mode, handle, mode;
```

```
set_mode = vswr_mode(handle, mode);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 32</code>	<code>intin(0) = mode</code>	<code>intout(0) = set_mode</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VS_COLOR (EH)

Set Color Representation assigns a specified color index to units of RGB (Red, Green, Blue) intensity. The calling routine specifies the unit of intensity for each of the three colors in tenths of a percent (0 - 1000).

If the specified value of a color intensity is less than 0, `vs_color` maps the value to 0; an intensity value of greater than 1000 is mapped to 1000.

On a monochrome device, the VDI maps any percentage of color to white. The background color is referenced as color index zero. The default color table index values are listed in Table 3-3 on page 3-4.

The number of color indices is device dependent; `work_out[13]` of `v_opnwk` (1H) and `v_opnvwk` (64H) returns the number of colors that can be defined. `v_opnwk` and `v_opnvwk` are described in Section 3, "Control Functions."

Note: This function performs no operation if a color lookup table is not supported for the specified device. `work_out[5]` of `vq_extnd` (66H), described in Section 8, indicates the availability of a color lookup table.

Input Arguments

<code>handle</code>	Device handle
<code>index</code>	The color index
<code>rgb_in[0]</code>	Red color intensity (in tenths of percent, 0-1000)
<code>rgb_in[1]</code>	Green color intensity (in tenths of percent, 0-1000)
<code>rgb_in[2]</code>	Blue color intensity (in tenths of percent, 0-1000)

Sample Call to C Language Binding

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

vs_color(handle, index, rgb_in);
```

Parameter Block Binding

Control	Input	Output
control(0) = 14	intin(0) = index	
control(1) = 0	intin(1) = rgb_in[0]	
control(2) = 0	intin(2) = rgb_in[1]	
control(3) = 4	intin(3) = rgb_in[2]	
control(4) = 0		
control(5) = 0		
control(6) = handle		

VSL_TYPE (FH)

This function sets the line type for subsequent polyline operations. All devices support at least six polyline types; the total number of available line types is device-dependent. `v_opnwk (1H)` and `v_opnwk (64H)` return the number of line types for a particular device in `work_out[6]`.

The line type index values and their corresponding pattern word bit settings are listed in Table 5-3. The pixel value in the pattern word is 1 = pixel on (active); 0 = pixel off, the most significant bit is the first pixel displayed.

Table 5-3. Line Type Index Values and Pattern Words

Value	Style	Line Pattern Word	
		MSB	LSB
1	Solid	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	FFFFH
2	Long dash	1 1 1 1 1 1 1 1 1 1 1 0 0 0 0	FFF0H
3	Dot	1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0	E0E0H
4	Dash,dot	1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0	FE38H
5	Dash	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0	FF00H
6	Dash,dot,dot	1 1 1 1 0 0 0 1 1 0 0 1 1 0 0 0	F198H
7	User-defined	See <code>vsl_udsty (71H)</code> on page 5-10	
8-n	Device-dependent		

By default at open workstation, the user-defined line style is set to the pattern word for a solid line. This default assignment can be changed with `vsl_udsty (71H)`.

`vsl_type` returns the line type selected and uses a solid line (1) if the value specified by the calling routine is out of range.

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. Check the information returned by `vq_extnd (66H)` in `work_out[17]` and `work_out[18]`; `vq_extnd` is described in Section 8, "Inquire Functions."

Input Arguments

handle Device handle
 style Requested line type, see Table 5-3

Output Arguments

set_type Line type selected

Sample Call to C Language Binding

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

set_type = vsl_type(handle, style);
```

Parameter Block Binding

Control	Input	Output
control(0) = 15	intin(0) = style	intout(0) = set_type
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VSL_UDSTY (71H)

`v_udsty` sets the user-defined line style pattern word in the device driver to the specified pattern word (16-bits). This line is used for subsequent polyline operations when an application selects the user-defined line style, index 7.

The Most Significant Bit (MSB) of the pattern word is the first pixel displayed in the line; see Table 5-3 on page 5-8.

When the workstation is opened, the default for the user-defined line style is a solid pattern word.

Input Arguments

<code>handle</code>	Device handle
<code>pattern</code>	Line style pattern word, 16 bits

Sample Call to C Language Binding

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

vsl_udsty(handle, pattern);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 113</code>	<code>intin(0) = pattern</code>	
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VSL_WIDTH (10H)

This function sets the line width 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 one. If you select two in Raster Coordinates, `vsl_width` returns one, which sets the line width at one pixel.

`vql_attributes` (23H), described in Section 8, "Inquire Functions," returns the current line width in `work_out[5]`.

Note: Thickened lines might be rendered on the device by selecting a solid line type (1).

Input Arguments

<code>handle</code>	Device handle
<code>width</code>	Requested line width in x-axis NDC/RC units

Output Arguments

<code>set_width</code>	Selected line width in x-axis NDC/RC units
------------------------	--

Sample Call to C Language Binding

```
WORD vsl_width();
WORD set_width, handle, width;

set_width = vsl_width(handle, width);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 16</code>	<code>ptsin(0) = width</code>	<code>ptsout(0) = set_width</code>
<code>control(1) = 1</code>	<code>ptsin(1) = 0</code>	<code>ptsout(1) = 0</code>
<code>control(2) = 1</code>		
<code>control(3) = 0</code>		
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VSL_COLOR (11H)

`vsl_color` sets the color index for subsequent polyline operations. The `vs_color` (EH) function determines the color the index represents.

At least two color indices, 0 and 1, are always supported (monochrome). Color indices range from 0 to a device-dependent maximum. `vsl_color` returns the color index selected and uses color index 1 if the calling process specifies an index that is out of range.

The default color table index values are listed in Table 3-3 on page 3-4.

Input Arguments

<code>handle</code>	Device handle
<code>color_index</code>	Requested color index

Output Arguments

<code>set_color</code>	Color index selected
------------------------	----------------------

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 17</code>	<code>intin(0) = color_index</code>	<code>intout(0) = set_color</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 5</code>		
<code>control(6) = handle</code>		

VSL_ENDS (6CH)

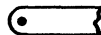
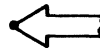
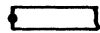
This function sets the style for the beginning and ending points of a polyline. The two points of a polyline can have different styles. The calling routine can specify the end styles listed in Table 5-4.

Table 5-4. Polyline End Styles

Value	End Style
0	Squared (Default)
1	Arrow
2	Rounded

If the calling routine requests an invalid end style, `vsl_ends` uses squared (0).

Both the squared and arrow styles end at the end of the polyline. The rounded style is drawn so that the center of the rounding is at the end of the polyline; see Figure 5-1.

**Figure 5-1. Points for Polyline End Styles**

Input Arguments

handle Device handle
beg_style End style for beginning point, see Table 5-4
end_style End style for ending point

Sample Call to C Language Binding

```

WORD  vsl_ends();
WORD  handle, beg_style, end_style;

    vsl_ends(handle, beg_style, end_style);
  
```

Parameter Block Binding

Control	Input	Output
control(0) = 108	intin(0) = beg_style	
control(1) = 0	intin(1) = end_style	
control(2) = 0		
control(3) = 2		
control(4) = 0		
control(5) = 0		
control(6) = handle		

VSM_TYPE (12H)

`vsm_type` sets the marker type for subsequent polymarker functions. Although the total number of available marker types is device-dependent, the VDI always defines at least six. The marker types are listed in Table 5-5.

Table 5-5. Polymarker Types

Value	Type
1	Dot
2	Plus
3	Asterisk (Default)
4	Square
5	Diagonal Cross
6	Diamond
7-n	User-defined

`v_opnwk` (1H) and `v_opnvwk` (64H) return the number of marker types for a particular device in `work_out[8]`; see Section 3. `vqm_attributes` (24H), described in Section 8, "Inquire Functions," returns the current marker type.

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

Input Arguments

handle Device handle
symbol Requested polymarker type, see Table 5-5

Output Arguments

set_type Polymarker type selected

Sample Call to C Language Binding

```
WORD vsm_type();
WORD set_type, handle, symbol;

set_type = vsm_type(handle, symbol);
```

Parameter Block Binding

Control	Input	Output
control(0) = 18	intin(0) = symbol	intout(0) = set_type
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VSM_HEIGHT (13H)

The polymarker height determined by `vsm_height` is used for subsequent polymarker functions. The number of polymarker sizes available for a particular device is returned by `v_opnwk (1H)` and `v_opnvwk (64K)` in `work_out[9]` -- see Section 3.

If the marker height requested by the calling routine does not exist, `vsm_height` selects the next smaller height. `vsm_height` returns the actual marker height selected in `set_height`.

Input Arguments

<code>handle</code>	Device handle
<code>height</code>	Requested polymarker height in y-axis NDC/RC units

Output Arguments

<code>set_height</code>	Polymarker height selected in y-axis NDC/RC units
-------------------------	---

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 19</code>	<code>ptsin(0) = 0</code>	<code>ptsout(0) = 0</code>
<code>control(1) = 1</code>	<code>ptsin(1) = height</code>	<code>ptsout(1) = set_height</code>
<code>control(2) = 1</code>		
<code>control(3) = 0</code>		
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VSM_COLOR (14H)

vsm_color sets the color index for subsequent polymarker operations. The vs_color (EH) function determines the color the index represents.

At least two color indices, 0 and 1, are always supported (monochrome). Color indices range from 0 to a device-dependent maximum. vsm_color returns the color index selected for the polymarker and selects color index 1 if the index specified by the calling process is out of range.

The default color table index values are listed in Table 3-3 on page 3-4.

Input Arguments

handle Device handle
color_index Requested polymarker color index

Output Arguments

set_color Selected polymarker color index

Sample Call to C Language Binding

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

set_color = vsm_color(handle, color_index);
```

Parameter Block Binding

Control	Input	Output
control(0) = 20	intin(0) = color_index	intout(0) = set_color
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VST_HEIGHT (CH)

`vst_height` sets the current graphic text character height in NDC/RC units (absolute mode). The character height is specified according to the distance from the character baseline to the top of the character cell, rather than the character cell height.

If the specified character height does not map exactly to a size available on device, `vst_height` selects the closest character size that does not exceed the requested size.

`vst_height` returns the selected character height (baseline to top line) and the size of a character cell; see Figure 5-2. For fixed (monospaced) fonts, `vst_height` returns the width of a character and the width of a character cell. For proportional fonts, `vst_height` returns the width of the widest character and the width of the widest character cell.

All input and output arguments are specified in units of the current coordinate system (NDC or RC).

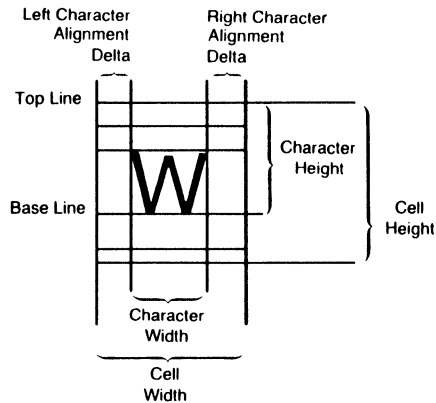


Figure 5-2. Character Size Definition

`v_opnwk` (1H), `v_opnvwk` (64H), and `vqt_attributes` (26H) return information about character sizes for the specified device. `v_opnwk` and `v_opnvwk` are described in Section 3; `vqt_attributes` is described in Section 8.

Input Arguments

<code>handle</code>	Device handle
<code>height</code>	Requested character height in y-axis NDC/RC units

Output Arguments

<code>char_width</code>	Character width selected in x-axis NDC/RC units
<code>char_height</code>	Character height selected in y-axis NDC/RC units
<code>cell_width</code>	Character cell width in x-axis NDC/RC units
<code>cell_height</code>	Character cell height in y-axis NDC/RC units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 12</code>	<code>ptsin(0) = 0</code>	<code>ptsout(0) = char_width</code>
<code>control(1) = 1</code>	<code>ptsin(1) = height</code>	<code>ptsin(1) = char_height</code>
<code>control(2) = 2</code>		<code>ptsout(2) = cell_width</code>
<code>control(3) = 0</code>		<code>ptsout(3) = cell_height</code>
<code>control(4) = 0</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VST_POINT (6BH)

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

If the character height specified by the calling routine does not map exactly to a device size, `vst_point` selects the closest character size that does not exceed the requested size.

`vst_point` returns the selected point size in `set_point`. Note that `vst_point` returns the character height, character width, cell height, and the cell width in NDC/RC units; see Figure 5-2 on page 5-19.

For proportional faces, `vst_point` returns the width of the widest character and the width of the widest character cell in the font.

`v_opnwk` (1H), `v_opnvwk` (64H), and `vqt_attributes` (26H) return information about character sizes for the specified device. `v_opnwk` and `v_opnvwk` are described in Section 3; `vqt_attributes` is described in Section 8.

Input Arguments

<code>handle</code>	Device handle
<code>point</code>	Character cell height in points

Output Arguments

<code>set_point</code>	Selected cell height in points
<code>char_width</code>	Selected character width in x-axis NDC/RC units
<code>char_height</code>	Selected character height in y-axis NDC/RC units
<code>cell_width</code>	Character cell width in x-axis NDC/RC units
<code>cell_height</code>	Character cell height in y-axis NDC/RC units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 107	intin(0) = point	intout(0) = set_point
control(1) = 0		ptsout(0) = char_width
control(2) = 2		ptsout(1) = char_height
control(3) = 1		ptsout(2) = cell_width
control(4) = 1		ptsout(3) = cell_height
control(5) = 0		
control(6) = handle		

VST_ROTATION (DH)

`vst_rotation` specifies the baseline for subsequent graphic text output by requesting an angle of rotation for the character baseline vector. The calling routine specifies the angle for the baseline vector in tenths of degrees. Figure 5-3 depicts how angles are specified to `vst_rotation`.

`vst_rotation` returns the selected baseline vector, which is a best-fit match to the requested value.

The default angle of rotation for the baseline vector is zero degrees when the workstation is opened.

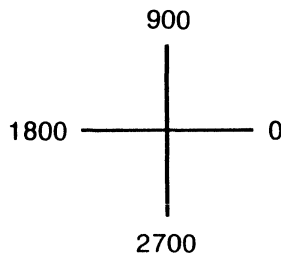


Figure 5-3. Angle Specification

Input Arguments

<code>handle</code>	Device handle
<code>angle</code>	Angle character baseline rotation (in tenths of degrees, 0 - 3600)

Output Arguments

<code>set_baseline</code>	Selected angle of character baseline rotation (in tenths of degrees 0-3600)
---------------------------	---

Sample Call to C Language Binding

```
WORD  vst_rotation();  
WORD  set_baseline, handle, angle;  
  
set_baseline = vst_rotation(handle, angle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 13	intin(0) = angle	intout(0) = set_baseline
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VST_FONT (15H)

This function selects a graphic character face for subsequent graphic text operations. The calling routine selects a face according to its number as listed in Table 5-6. You can load external faces with `v_load_fonts` (77H), described Section 3.

Note that some faces might not be supported on all devices. Use the Inquire Font Name and Index function, `vqt_name` (82H), to determine face names and indices. `vqt_attributes` (26H) returns the currently selected face in `work_out[0]`; `vqt_name` and `vqt_attributes` are described in Section 8, "Inquire Functions." `v_opnwk` (1H) and `v_opnvwk` (64H) both return the number of faces for the device in `work_out[10]` -- see Section 3.

Table 5-6. Face Values and Names

Value	Font
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

Input Arguments

handle Device handle

font Number of requested software text face

Output Arguments

set_font Number of face selected

Sample Call to C Language Binding

```
WORD vst_font();
WORD set_font, handle, font;

set_font = vst_font(handle, font);
```

Parameter Block Binding

Control	Input	Output
control(0) = 21	intin(0) = font	intout[0] = set_font
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VST_COLOR (16H)

This function sets the color index for subsequent graphic text operations. The vs_color (EH) function determines the color the index represents.

At least two color indices, 0 and 1, are always supported (monochrome). Color indices range from 0 to a device-dependent maximum. vst_color returns the color index selected and uses color index 1 if the calling process specifies an index that is out of range.

The default color table index values are listed in Table 3-3 on page 3-4.

Input Arguments

handle Device handle
color_index Requested text color index

Output Arguments

set_color Selected text color index

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 22	intin(0) = color_index	intout(0) = set_color
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VST_EFFECTS (6AH)

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

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

The **effect** input argument is a bit pattern. The attributes set by `vst_effects` correspond to the setting in the six least significant bits. Table 5-7 lists the bit assignments.

Table 5-7. Text Attribute Bit Mapping

Bit	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 underline text
4	Outline 0 do not outline 1 outline
5	Shadow 0 no shadow 1 shadow

If effect = 9 (1001 binary), for example, the text style is set to thickened and underlined.

For effects not supported on a device, `vst_effects` returns those bits set to 0 in `set_effect`. `work_out[2]` of `vq_extnd` (66H) returns the supported text effects.

The default text effect when the workstation is opened is normal intensity.

Figure 5-4 depicts the five of the special text effects.

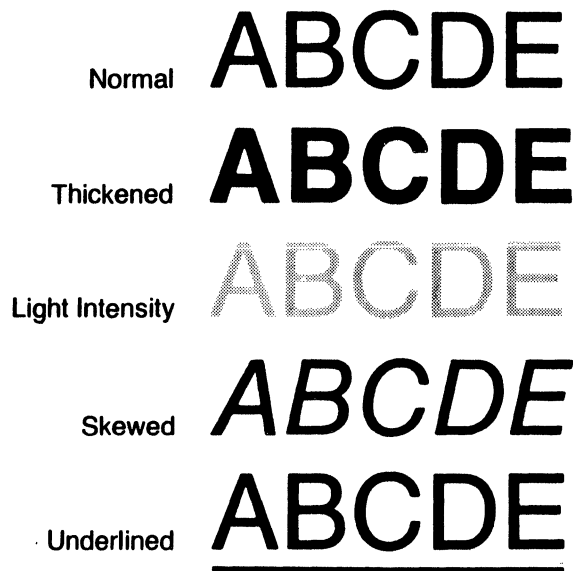


Figure 5-4. Graphic Text Special Effects

Input Arguments

handle	Device handle
effect	Text effect bit pattern

Output Arguments

set_effect Text effects actually selected (text effect word with the appropriate bits set)

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 106	intin(0) = effect	intout(0) = set_effect
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VST_ALIGNMENT (27H)

vst_alignment sets horizontal and vertical alignment for graphic text. The horizontal alignment is the direction of the baseline; the vertical alignment is perpendicular to the baseline. This function controls the positioning of the text string in relation to the graphic text position.

The default alignment, when the workstation is opened, places the left baseline corner of the string at the graphic text position.

If the calling routine specifies an invalid horizontal alignment, vst_alignment selects the default, left justified. vst_alignment selects the default vertical alignment, baseline, if the calling routine specifies an invalid vertical alignment.

vqt_attributes (26H) returns the current horizontal and vertical alignment settings in work_out[3] and work_out[4], respectively; see Section 8, "Inquire Functions."

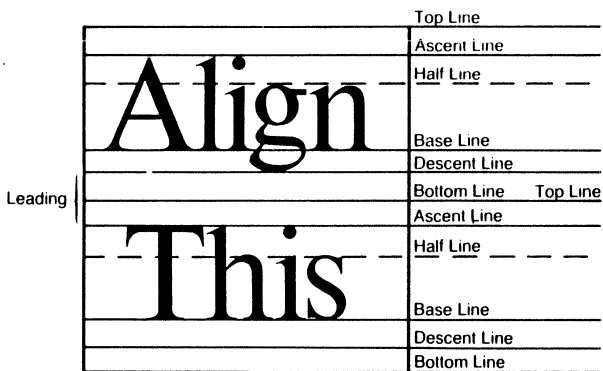


Figure 5-5. Graphic Text Alignment

Input Arguments

handle	Device handle
hor_in	Horizontal alignment: <ul style="list-style-type: none"> 0 Left justified (default) 1 Center justified 2 Right justified
vert_in	Vertical alignment (see Figure 5-5): <ul style="list-style-type: none"> 0 Baseline (default) 1 Half line 2 Ascent line 3 Bottom line 4 Descent line 5 Top line

Output Arguments

hor_out	Selected horizontal alignment
vert_out	Selected vertical alignment

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 39	intin(0) = hor_in	intout(0) = hor_out
control(1) = 0	intin(1) = vert_in	intout(1) = vert_out
control(2) = 0		
control(3) = 2		
control(4) = 2		
control(5) = 0		
control(6) = handle		

VSF_INTERIOR (17H)

The fill interior style to be used in subsequent polygon fill operations is determined by `vsf_interior`.

`vsf_interior` returns the selected fill interior style. If the calling routine specifies an unavailable style, `vsf_interior` selects hollow. Hollow style fills the interior with the current background color (color index 0). Solid style fills the area with the currently selected fill color.

`vqf_attributes` (25H), described in Section 8, returns both the current fill interior style and fill style index values.

Input Arguments

<code>handle</code>	Device handle
<code>style</code>	Fill interior style:
	0 Hollow
	1 Solid
	2 Pattern
	3 Hatch
	4 User-defined style

Output Arguments

`set_interior` Selected fill interior style

Sample Call to C Language Binding

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

Parameter Block Binding

Control	Input	Output
control(0) = 23	intin(0) = style	intout(0) = set_interior
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VSF_STYLE (18H)

`vsf_style` selects a fill style based on the fill interior style determined by `vsf_interior` (17H). If the current fill interior style is hollow (0), solid (1), or user-defined (4), this function has no effect on the fill style. A fill style index can be selected only for the pattern (2) and hatch (3) fill interior styles.

Fill style indices range from 1 to a device-dependent maximum. `work_out[11]` and `work_out[12]` of `v_opnwk` (1H) return this maximum; see Section 3. If the calling routine specifies an unavailable fill style index, `vsf_style` uses index 1.

Figure 5-6 shows the available fill styles. Under each fill style shown Figure 5-6 are two numbers separated by a comma. The number to the left of the comma corresponds to the interior style: Hollow, Pattern, or Hatch. The number to the right of the comma corresponds to the index for the particular pattern or hatch.

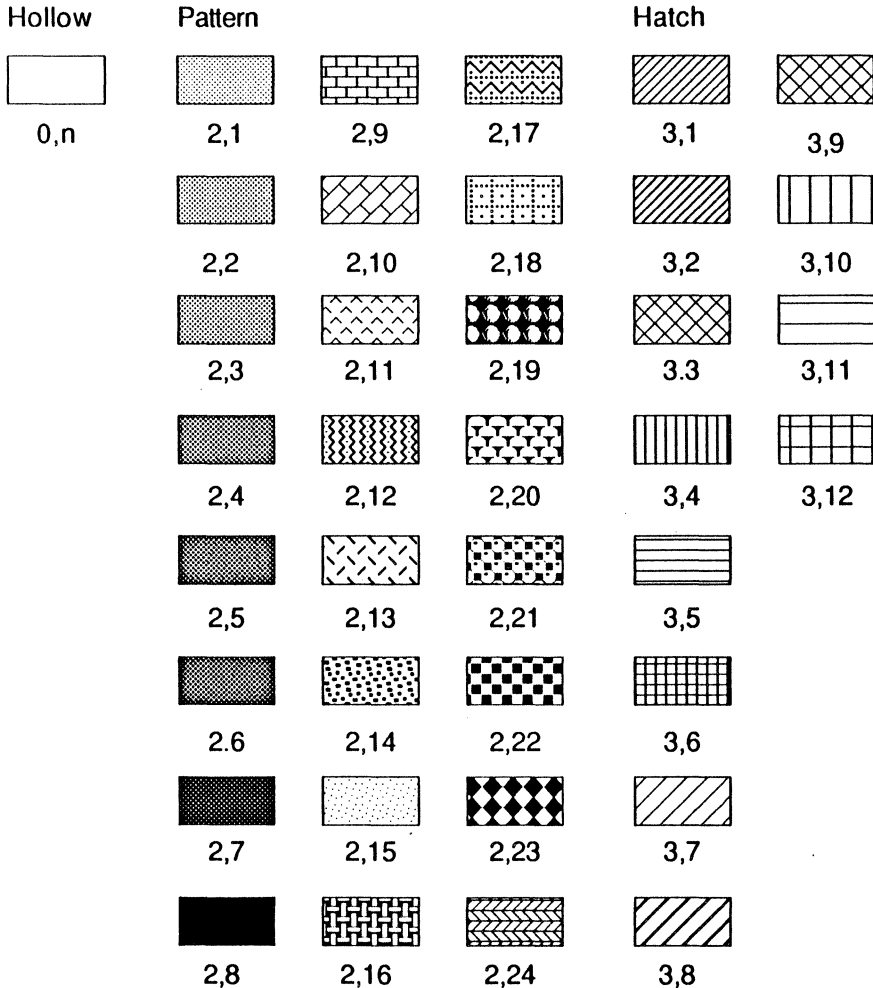


Figure 5-6. Fill Styles and Indices

Note: 1,n (interior style 1, solid, 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.

The number of available pattern and hatch styles is returned in `work_out[11]` and `work_out[12]`, respectively, of `v_opnwk` (1H) and `v_opnvwk` (64H). These functions are described in Section 3.

Input Arguments

`handle` Device handle
`style_index_Fill` style index for pattern or hatch interior

Output Arguments

`set_style` Selected fill style index for pattern or hatch interior

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 24</code>	<code>intin(0) = style_index</code>	<code>intout(0) = set_style</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VSF_COLOR (19H)

`vsf_color` sets the color index for subsequent fill operations. Set Color Representation, `vs_color` (EH), determines the color the index represents.

At least two color indices, 0 and 1, are always supported (monochrome). Color indices range from 0 to a device-dependent maximum. `vsf_color` returns the color index selected and uses color index 1 if the calling process specifies an invalid index.

The default color table index values are listed in Table 3-3 on page 3-4.

Input Arguments

<code>handle</code>	Device handle
<code>color_index</code>	Requested fill color index

Output Arguments

<code>set_color</code>	Selected fill color index
------------------------	---------------------------

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
<code>control(0) = 25</code>	<code>intin(0) = color_index</code>	<code>intout(0) = set_color</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VSF_PERIMETER (68H)

vsf_perimeter turns the outline of a fill area on or off. When perimeter visibility is on (the default when the workstation is opened), the border of a fill area is drawn in the current fill area color with a solid line. When perimeter visibility is off, no outline is drawn.

Any nonzero value specified for the perimeter visibility flag, per_vis, causes the perimeter to be visible.

Input Arguments

handle	Device handle
per_vis	Perimeter visibility flag:
zero	Invisible
nonzero	Visible

Output Arguments

set_perimeter	Selected perimeter visibility
---------------	-------------------------------

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 104	intin(0) = per_vis	intout(0) = set_perimeter
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VSF_UDPAT (70H)

This function redefines the user-definable fill pattern. This fill pattern is referenced by `vsf_interior` (17H) as fill interior style 4.

16 two-byte words are required for a single plane fill pattern. Bit 15 of the first word is the upper left bit of the pattern. Bit 0 of word 16 is the lower right bit of the pattern. Bit 0 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 as set by `vsf_color` (19H).

For a multiple plane pattern (multiple colors), the number of full 16-by-16 (words-by-bits) planes defined is used in the fill operation. Any unspecified planes are zeroed. Note that the writing mode must be set to replace (mode 1) when using a multiplane fill pattern; see `vswr_mode` (20H) on page 5-3.

The user-defined fill pattern is the DRI logo by default when the workstation is opened.

Input Arguments

<code>handle</code>	Device handle
<code>pfill_pat</code>	Fill pattern data (16 words per plane)

Sample Call to C Language Binding

```
WORD vsf_udpat();  
WORD handle, planes, pfill_pat[planes x 16];  
  
vsf_udpat(handle, pfill_pat, planes);
```

Parameter Block Binding

Control	Input
control(0) = 112	intin(0)
control(1) = 0	.
control(2) = 0	.
control(3) = 16-n	intin(15) = 1st plane of fill pattern
control(4) = 0	intin(16)
control(5) = 0	.
control(6) = handle	.
	intin(29) = 2nd plane of fill pattern
	intin(n-15)
	.
	.
	intin(n) = last plane of fill pattern

End of Section 5

Raster Operations

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

Table 6-1. Raster Operation Functions

Function	Page	Purpose
v_get_pixel (69H)	6-9	Get Pixel
vro_cpyfm (6DH)	6-10	Copy raster, opaque
vr_trnfm (6EH)	6-12	Transform form
vrt_cpyfm (79H)	6-13	Copy raster, transparent

Memory Form Definition Block

A raster area is defined by a Memory Form Definition Block (MFDB). As shown in Figure 6-1, an MFDB consists of ten words; its components are as follows:

- A 32-bit pointer to the memory address of the upper left corner of the first plane of the raster area. This pointer is a native-machine pointer. If all 32 bits of this pointer are 0, the MFDB is for a physical device, and any MFDB parameters you enter are ignored -- these values are defined in the device driver.
- The width and height of the raster area in pixels.
- 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.
- A flag indicating whether the format of the raster area is standard or device-dependent.
- The number of memory planes in the raster area.
- 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.

————— One word (16 bits) —————

Word 1	Memory pointer word 1
Word 2	Memory pointer word 2
Word 3	Form Width in Pixels
Word 4	Form Height in Pixels
Word 5	Form Width in Words
Word 6	Form format flag
Word 7	Number of Memory Planes
Word 8	Reserved for future use
Word 9	Reserved for future use
Word 10	Reserved for future use

Figure 6-1. Memory Form Definition Block

Raster Area Formats

Two memory formats are associated with raster areas:

- device-specific format
- well-defined standard format

The VDI provides the `vr_trnfm` (6EH) function to transform a raster area from one format to another. You must transform a standard format raster area to a device-specific format before using a copy raster function, `vro_cpyfm` (6DH) or `vrt_cpyfm` (79H).

The form format flag of the MFDB 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 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 6-2 and 6-3 define the pixel-value-to-color-index mapping for eight-color and sixteen-color screens, respectively.
- The 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. The first word is on the left edge of the row.

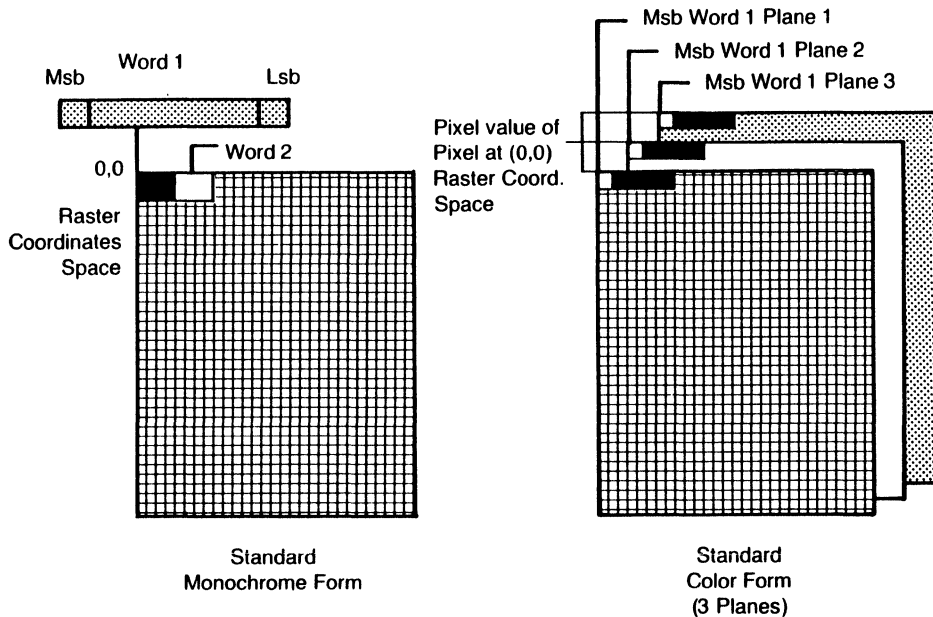


Figure 6-2. Standard Forms

Table 6-2. 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 6-3. 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	Light Grey
1000	9	Dark Grey
1001	10	Dark Red
1010	11	Dark Green
1011	14	Dark Yellow
1100	12	Dark Blue
1101	15	Dark Magenta
1110	13	Dark Cyan
1111	1	Black

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

In addition to the MFDB, the copy raster functions `vro_cpyfm` (6DH) and `VRT_cpyfm` (79H) also take 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 two diagonal corners.

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 (5,4) is shown in Figure 6-3.

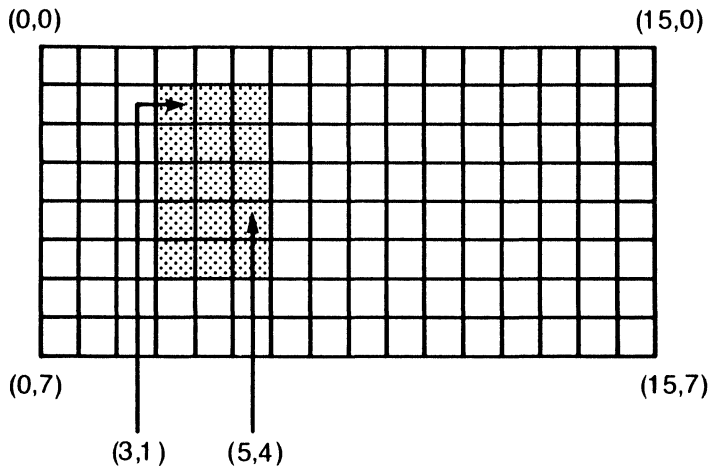


Figure 6-3. Sample Single Plane Memory Form

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 6-4 lists the available operations with the following conventions:

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

Table 6-4. Raster Operation Logic Operations

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

V_GET_PIXEL (69H)

This function returns a pixel value and a color index for the pixel specified in *x* and *y* by the calling routine.

Color index 0 is the background color. It might not map to pixel value 0 in device-specific form. Refer to Tables 6-2 and 6-3 for the colors and values. Standard form always maps color index 0 to pixel value 0.

Note: `v_get_pixel` may not be supported by all device drivers.

Input Arguments

<code>handle</code>	Device handle
<code>x</code>	<i>x</i> -coordinate of pixel in RC/NDC units
<code>y</code>	<i>y</i> -coordinate of pixel in RC/NDC units

Output Arguments

<code>pel</code>	Pixel value in device-specific form
<code>index</code>	Color index

Sample Call to C Language Binding

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

v_get_pixel(handle, x, y, &pel, &index);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 105</code>	<code>ptsin(0) = x</code>	<code>intout(0) = pel</code>
<code>control(1) = 1</code>	<code>ptsin(1) = y</code>	<code>intout(1) = index</code>
<code>control(2) = 0</code>		
<code>control(3) = 0</code>		
<code>control(4) = 2</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VRO_CPYFM (6DH)

Copy Raster Opaque copies a rectangular raster area from source form to destination form using the logic operation specified by the calling routine. If the source and destination forms are the same, and the rectangles overlap, vro_cpyfm copies so that the source rectangle is not changed until the corresponding area in the destination has been processed. 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, vro_cpyfm uses the destination as a pointer and the source for the size. vq_extnd (66H), described in Section 8, returns scaling ability.

The source and destination forms must be in device-specific form; see vr_trnfm (6EH) in this section.

All input coordinates (xy[0] to xy[7]) must be specified according to the current coordinate system (NDC or RC).

Input Arguments

handle	Device handle
srcMFDB	Double-word address of the source MFDB
desMFDB	Double-word address of the destination MFDB
wr_mode	Logic operation (refer to Table 6-4)
xy[0]	x-coordinate of source rectangle corner in NDC/RC
xy[1]	y-coordinate of source rectangle corner in NDC/RC
xy[2]	x-coordinate, in NDC/RC, of corner diagonal to xy[0]
xy[3]	y-coordinate, in NDC/RC, of corner diagonal to xy[1]
xy[4]	x-coordinate of destination rectangle corner in NDC/RC
xy[5]	y-coordinate of destination rectangle corner in NDC/RC

xy[6] x-coordinate of destination rectangle corner diagonal
to xy[4]

xy[7] y-coordinate of destination rectangle corner diagonal
to xy[5]

Sample Call to C Language Binding

```
WORD vro_cpyfm();
WORD handle, wr_mode, xy[], *srcMFDB, *desMFDB;

vro_cpyfm(handle, wr_mode, xy, srcMFDB, desMFDB);
```

Parameter Block Binding

Control	Input	Output
control(0) = 109	intin(0) = wr_mode	
control(1) = 4	ptsin(0) = xy[0]	
control(2) = 0	ptsin(1) = xy[1]	
control(3) = 1	.	
control(4) = 0	.	
control(5) = 0	ptsin(7) = xy[7]	
control(6) = handle		
control(7-8) = srcMFDB		
control(9-10) = desMFDB		

VR_TRNFM (6EH)

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 that changes the current format 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.

Note that the source and destination MFDBs must be either completely coincident or separate; they cannot partially overlap.

Input Arguments

handle	Device handle
srcMFDB	Double-word address of the source MFDB
desMFDB	Double-word address of the destination MFDB

Sample Call to C Language Binding

```
WORD vr_trnfm();
WORD handle, *srcMFDB, *desMFDB;

vr_trnfm(handle, srcMFDB, desMFDB);
```

Parameter Block Binding

Control	Input	Output
control(0) = 110		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		
control(7-8) = srcMFDB		
control(9-10) = desMFDB		

VRT_CPYFM (79H)

Copy Raster Transparent copies a monochrome rectangular raster area from the source form to a color area. The calling routine specifies a writing mode and color indices for both 0's and 1's. The source form cannot be the screen.

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

All rectangle coordinates (`xy[0]` to `xy[1]`) must be specified according to the current coordinate system (NDC or RC).

Transfer of information from the source to the destination is controlled by specifying the writing mode as described below.

Mode 1, Replace

Replace mode results in a replacement of all pixels in the destination rectangle. The foreground color index, specified in `index[0]`, is output to all pixels associated with source locations which are set to one. The background color index, specified in `index[1]`, is output to all pixels associated with source locations set to zero.

Mode 2, Transparent

Transparent mode affects only the pixels associated with a source value of one. Those pixels are set to the foreground color whose index is specified in `index[0]`. The color index specified in `index[1]` is not used.

Mode 3, XOR

In XOR mode, the monochrome raster source area is logically XORed with each plane of the destination. The specified color indices are not used.

Mode 4, Reverse Transparent

Reverse Transparent mode affects only the pixels associated with a source value of zero. These pixels are set to the background color specified in index[1]. The color index specified in index[0] is not used.

Input Arguments

handle	Device handle
srcMFDB	Double-word address of the source MFDB
desMFDB	Double-word address of the destination MFDB
wr_mode	Writing Mode: 1 Replace mode 2 Transparent mode 3 XOR mode 4 Reverse transparent mode
index[0]	Color index for ones in data (foreground)
index[1]	Color index for zeros in data (background)
xy[0]	x-coordinate of source rectangle corner in NDC/RC
xy[1]	y-coordinate of source rectangle corner in NDC/RC
xy[2]	x-coordinate, in NDC/RC, of corner diagonal to xy[0]
xy[3]	y-coordinate, in NDC/RC, of corner diagonal to xy[1]
xy[4]	x-coordinate of destination rectangle corner in NDC/RC
xy[5]	y-coordinate of destination rectangle corner in NDC/RC
xy[6]	x-coordinate, in NDC/RC, of corner diagonal to xy[4]
xy[7]	y-coordinate, in NDC/RC, of corner diagonal to xy[5]

Sample Call to C Language Binding

```
WORD vrt_cpyfm();
WORD handle, wr_mode, xy[], *srcMFDB *desMFDB, index;

vrt_cpyfm(handle, wr_mode, xy, srcMFDB, desMFDB, &index);
```

Parameter Block Binding

Control	Input	Output
control(0) = 121	intin(0) = wr_mode	
control(1) = 4	intin(1) = index[0]	
control(2) = 0	intin(2) = index[1]	
control(3) = 3	ptsin(0) = xy[0]	
control(4) = 0	ptsin(1) = xy[1]	
control(5) = 0	.	
control(6) = handle	.	
control(7-8) = srcMFDB	ptsin(7) = xy[7]	
control(9-10) = desMFDB		

End of Section 6

Input Functions

The VDI 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.

In VDI-only programs with multiple workstations, use the request mode rather than the sample mode to input characters. Workstations using the sample mode might not get all of the characters input.

Note: Programs that use both the Application Environment Services (AES) and VDI functions must use the AES input functions rather than the VDI functions. Use the VDI input functions only in programs that do not use the AES. The AES is described in the GEM Application Environment Services Reference Guide.

Table 7-1 lists the VDI input functions.

Table 7-1. VDI Input Functions

Function	Page	Purpose
vrq_locator (1CH)	7-5	Input locator, request mode
vsm_locator (1CH)	7-7	Input locator, sample mode
vrq_valuator (1DH)	7-9	Input valuator, request mode
vsm_valuator (1DH)	7-11	Input valuator, sample mode
vrq_choice (1EH)	7-13	Input choice, request mode
vsm_choice (1EH)	7-14	Input choice, sample mode
vrq_string (1FH)	7-16	Input string, request mode
vsm_string (1FH)	7-18	Input string, sample mode
vsin_mode (21H)	7-3	Set input mode
vsc_form (6FH)	7-20	Set mouse form
vex_timv (76H)	7-22	Exchange timer interrupt vector
v_show_c (7AH)	7-24	Show cursor
v_hide_c (7BH)	7-26	Hide cursor
vq_mouse (7CH)	7-27	Sample mouse button state
vex_butv (7DH)	7-28	Exchange button change vector
vex_motv (7EH)	7-30	Exchange mouse movement vector
vex_curv (7FH)	7-32	Exchange cursor change vector
vq_key_s (80H)	7-34	Sample keyboard state information

VSIN_MODE (21H)

`vsin_mode` sets the input mode to request or sample for the specified logical input device. The logical input devices are:

- locator (mouse, trackball, or joystick)
- valuator (potentiometer)
- choice (function keys)
- string (keyboard)

Input Arguments

`handle` Device handle

`dev_type` Logical input device:

- 1 Locator
- 2 Valuator
- 3 Choice
- 4 String

`mode` Input mode:

- 1 Request
Driver does not return until an input event occurs
- 2 Sample
Driver returns current status or location of the input device without waiting

Sample Call to C Language Binding

WORD `vsin_mode()`;

WORD `handle, dev_type, mode;`

`vsin_mode(handle, dev_type, mode);`

Parameter Block Binding

Control	Input	Output
control(0) = 33	intin(0) = dev_type	intout(0) = Selected mode
control(1) = 0	intin(1) = mode	
control(2) = 0		
control(3) = 2		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VRQ_LOCATOR (1CH)

This function returns the position of the specified locator device in request mode. Upon entry to the locator routine, the current cursor form is displayed at the initial coordinate. Because `vrq_locator` operates in request mode, it tracks the graphic cursor with the input device until a terminating event occurs, which can result from the user pressing a key or a button on a mouse. `vrq_locator` 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. If both a keyboard and another locator device are available, the cursor is tracked by input from either. This gives the user maximum flexibility.

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

Note: This function is not required and might not be supported by all device drivers.

Input Arguments

<code>handle</code>	Device handle
<code>initx</code>	Initial x-coordinate of locator in NDC/RC units
<code>inity</code>	Initial y-coordinate of locator in NDC/RC units

Output Arguments

xout	Final x-coordinate of locator in NDC/RC units
yout	Final y-coordinate of locator in NDC/RC units
term	Locator terminator The low byte contains a character terminator. For keyboard-terminated locator input, this is the ASCII character code of the key that was struck to terminate input. For nonkeyboard-terminated input (tablet or mouse for example), valid locator terminators begin with 20H (space) and increase from there. For instance, if the puck on a tablet has 4 buttons, the first button must generate 20H as a terminator, the second 21H, the third 22H, and the fourth 23H.

Sample Call to C Language Binding

```
WORD vrq_locator();
WORD handle, initx, inity, xout, yout, term;

vrq_locator(handle, initx, inity, &xout, &yout, &term);
```

Parameter Block Binding

Control	Input	Output
control(0) = 28	ptsin(0) = initx	intout(0) = term
control(1) = 1	ptsin(1) = inity	ptsout(0) = xout
control(2) = 1		ptsout(1) = yout
control(3) = 0		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VSM_LOCATOR (1CH)

vsm_locator returns the position of the specified locator device. Input is sampled. Upon entry to the locator routine, no cursor is displayed. Use v_show_c (7AH), described on page 7-24, to display the cursor. If the cursor position has changed, vsm_locator returns the cursor position; if a terminating event occurred, vsm_locator returns a character. See Table 7-2 for the appropriate settings of control(2) and control(4).

Note: This function is not required and might not be supported by all device drivers. If both a keyboard and another locator device are available, the input comes from either, giving the user maximum flexibility.

Input Arguments

handle	Device handle
initx	Initial x-coordinate of locator in NDC/RC units
inity	Initial y-coordinate of locator in NDC/RC units

Output Arguments

xout	New x-coordinate of locator in NDC/RC units
yout	New y-coordinate of locator in NDC/RC units
term	Locator keypress if keypress occurs This information is the same as for vrq_locator (1CH), see page 7-5.

Table 7-2. Sample Mode Status Returned

Event	Control(2)	Control(4)
Coordinates changed	1	0
Key pressed; coordinates unchanged	0	1
No input	0	0
Key pressed; coordinates changed	1	1

Sample Call to C Language Binding

```
WORD vsm_locator();
WORD handle, initx, inity, xout, yout, term;

vsm_locator(handle, initx, inity, &xout, &yout, &term);
```

Parameter Block Binding

Control	Input	Output
control(0) = 28	ptsin(0) = x	intout(0) = term
control(1) = 1	ptsin(1) = y	ptsout(0) = xout
control(2) = 1 coordinate changed 0 no coordinate changed		ptsout(1) = yout
control(3) = 0		
control(4) = 0 no keypress character 1 keypress character returned		
control(5) = 0		
control(6) = handle		

VRQ_VALUATOR (1DH)

This function returns the value of the valuator device. Because it operates in request mode, `vrq_valuator` increments or decrements the initial value of the valuator 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: `vrq_valuator` is not required and might not be supported by all device drivers.

Input Arguments

<code>handle</code>	Device handle
<code>val_in</code>	Initial value

Output Arguments

<code>val_out</code>	Output value
<code>term</code>	Terminator

Sample Call to C Language Binding

```
WORD  vrq_valuator();  
WORD  handle, val_in, val_out, term;  
  
vrq_valuator(handle, val_in, &val_out, &term);
```

Parameter Block Binding

Control	Input	Output
control(0) = 29	intin(0) = val_in	intout(0) = val_out
control(1) = 0		intout(1) = term
control(2) = 0		
control(3) = 1		
control(4) = 2		
control(5) = 0		
control(6) = handle		

VSM_VALUATOR (1DH)

vsm_valuator returns the current value of the valuator device in sample mode. If the valuator has changed, vsm_valuator increments or decrements the valuator value as required. If a terminating event occurs, vsm_valuator returns the value. If nothing happens, the function does not return a value.

Valuator numbers range from 1 to 100. The suggested keys are the same as for vrq_valuator (1DH); see page 7-9.

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

Input Arguments

handle	Device handle
val_in	Initial value

Output Arguments

val_out	New valuator value
term	Keypress, if keypress event occurred
status	Terminating status:
	0 Nothing happened
	1 Valuator changed
	2 Keypress character

Sample Call to C Language Binding

```
WORD vsm_valuator();  
WORD handle, val_in, val_out, term, status;  
  
vsm_valuator(handle, val_in, &val_out, &term, &status);
```

Parameter Block Binding

Control	Input	Output
control(0) = 29	intin(0) = val_in	intout(0) = val_out
control(1) = 0		intout(1) = term
control(2) = 0		
control(3) = 1		
control(4) = status		
control(5) = 0		
control(6) = handle		

VRQ_CHOICE (1EH)

Operating in request mode, this function returns the status of the selected choice device (function key). `vrq_choice` samples input until a key is pressed; if a valid choice key is pressed, `vrq_choice` returns its value. Otherwise, `vrq_choice` returns the initial choice number.

Choice numbers range from one to a device-dependent maximum value. `v_opnwk (1H)` and `v_opnvwk` return the number of choice devices in `work_out[42]`; see Section 3.

Note: This function is not required and might not be supported by all device drivers.

Input Arguments

<code>handle</code>	Device handle
<code>in_choice</code>	Initial choice number

Output Arguments

<code>out_chioce</code>	Choice key number
-------------------------	-------------------

Sample Call to C Language Binding

```
WORD vrq_choice();
WORD handle, in_choice, out_choice;

vrq_choice(handle, in_choice, &out_choice);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 30</code>	<code>intin(0) = in_choice</code>	<code>intout(0) = out_choice</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VSM_CHOICE (1E)

vsm_choice returns the choice status of the selected choice device. Upon entry to the routine, vsm_choice samples input. If input is available and is a valid choice key, vsm_choice returns it. Choice numbers range from one to a device-dependent maximum value. v_opnwk (1H) and v_opnvwk return the number of choice devices in work_out[42]; see Section 3.

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

Input Arguments

handle	Device handle
--------	---------------

Output Arguments

choice	Choice number if sample successful, 0 if unsuccessful
status	Choice status:
	0 Nothing happened
	1 Sample successful

Sample Call to C Language Binding

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

vsm_choice(handle, &choice);
```

Parameter Block Binding

Control	Input	Output
control(0) = 30		intout(0) = choice
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = status		
control(5) = 0		
control(6) = handle		

VRQ_STRING (1FH)

Operating in request mode, `vrq_string` returns a string from the specified device. Input is accumulated until `vrq_string` encounters a carriage return or the `intout` array is full (maximum string length has been reached).

If the calling routine enables echo mode, text will be echoed to the screen with the current text attributes using the vertex passed in `echo_yx` as the justification point. Note that echo mode might not be available on all devices. `vqt_attributes`, described in Section 8, returns the current text attributes.

If the number passed as the maximum string length is negative, the output string values returned by `vrq_string` conform to the standard keyboard defined in Appendix D. In this case, the absolute value of length is used as the maximum string size.

Note: Echo mode might not be available on all devices.

Input Arguments

<code>handle</code>	Device handle
<code>length</code>	Maximum string length
<code>echo_mode</code>	Echo mode: 0 No echo 1 Echo input characters at specified position
<code>echo_xy</code>	x- and y-coordinates of echo area in NDC/RC units

Output Arguments

<code>string</code>	Output string returned in ADE
---------------------	-------------------------------

The string is word-oriented for the Parameter Block Binding. For the C Binding, the string is byte-oriented and null-terminated; its length includes an additional byte for the terminating null.

Sample Call to C Language Binding

```

WORD  vrq_string();
WORD  handle, length, echo_mode, echo_xy[];
BYTE  *string;

```

```
vrq_string(handle, length, echo_mode, echo_xy, string);
```

Parameter Block Binding

Control	Input	Output
control(0) = 31	intin(0) = length	intout = string
control(1) = 1	intin(1) = echo_mode	
control(2) = 0	ptsin(0) = echo_xy(0)	
control(3) = 2	ptsin(1) = echo_xy(1)	
control(4) = string length		
control(5) = 0		
control(6) = handle		

VSM_STRING (1FH)

`vsm_string` returns a string from the specified device. Upon entry, `vsm_string` samples input; if data is available, it is accumulated, and the function samples the input again. `vsm_string` accumulates input until one of the following events occurs:

- data is no longer available
- a carriage return is encountered
- the maximum string length is reached

If the string will always be terminated with RETURN, use `vrq_string` (1FH); see page 7-16.

If the calling routine enables echo mode, text will be echoed to the screen with the current text attributes using the vertex passed in `echo_yx` as the justification point. `vqt_attributes`, described in Section 8, returns the current text attributes.

Note: Echo mode might not be available on all devices.

If the number passed as the maximum string length is negative, the output string values returned by `vrq_string` conform to the standard keyboard defined in Appendix D. In this case, the absolute value of the length argument is used as the maximum string size.

Input Arguments

<code>handle</code>	device handle
<code>length</code>	Maximum string length
<code>echo_mode</code>	Echo mode: <ul style="list-style-type: none"> 0 No echo 1 Echo input characters at the specified position
<code>echo_xy</code>	x- and y-coordinates of echo area in NDC/RC units

Output Arguments

string	Output string, if sample successful The string is word-oriented for the Parameter Block Binding. For the C Binding, the string is byte-oriented and null-terminated; its length includes an additional byte for the terminating null.
status	Length of output string: 0 Sample unsuccessful (characters not available) >0 Sample successful (characters available)

Sample Call to C Language Binding

```
WORD  vsm_string();
WORD  status, handle, length, echo_mode, echo_xy[];
BYTE  *string;
```

```
vsm_string(handle, length, echo_mode, echo_xy, string, &status);
```

Parameter Block Binding

Control	Input	Output
control(0) = 31	intin(0) = length	intout = string
control(1) = 1	intin(1) = echo_mode	
control(2) = 0	ptsin(0) = echo_xy(0)	
control(3) = 2	ptsin(1) = echo_xy(1)	
control(4) = status		
control(5) = 0		
control(6) = handle		

VSC_FORM (6FH)

`vsc_form` redefines the cursor pattern displayed during locator input or at any time the cursor is shown (see the description of `v_show_c` (7AH) on page 7-24).

The size of the cursor is 16 by 16 pixels; it is defined by two arrays, cursor mask and cursor data. 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 hot spot determines the location of the cursor; its coordinates are specified in `cur_form[0]` and `cur_form[1]`.

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. 1's in the mask cause the corresponding pixel to be set to the color index designated in `cur_form[3]`.
3. 1's in the mouse form data cause the corresponding pixel to be set to the color index designated in `cur_form[4]`.

Input Arguments

handle	Device handle
cur_form[0]	x-coordinate of hot spot
cur_form[1]	y-coordinate of hot spot
cur_form[2]	Reserved for future use, must be 1
cur_form[3]	Mask color index, normally 0
cur_form[4]	Data color index, normally 1
cur_form[5]	
to	
cur_form[20]	16 words of 16-bit cursor mask
cur_form[21]	
to	
cur_form[36]	16 words of 16-bit cursor data

Sample Call to C Language Binding

```
WORD vsc_form();
WORD handle, cur_form[37];

vsc_form(handle, cur_form);
```

Parameter Block Binding

Control	Input	Output
control(0) = 111	intin(0) = cur_form[0]	
control(1) = 0	.	
control(2) = 0	.	
control(3) = 37	intin(36) = cur_form[36]	
control(4) = 0		
control(5) = 0		
control(6) = handle		

VEX_TIMV (76H)

`vex_timv` allows the application to perform some action each time a timer tick occurs.

Input to this function is a two-word pointer, `tim_addr`, that indicates the starting address of the application-dependent code that is to receive control when a timer tick occurs. `vex_timv` returns the address of the old timer routine in `old_addr` and the number of milliseconds per timer tick in `scale`.

The application-dependent code is invoked with a processor-dependent instruction. When this is complete, the application should perform a processor-dependent return instruction. See Appendix E for processor specific instructions and register names.

Note: 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.

This function is available only to the application that opened the physical workstation. The device handle must be the "root" or physical device handle.

Input Arguments

<code>handle</code>	Physical device handle
<code>tim_addr</code>	Address of application timer routine

Output Arguments

<code>old_addr</code>	Address of the old timer routine
<code>scale</code>	Milliseconds per tick

Sample Call to C Language Binding

```
WORD vex_timv();  
WORD handle, scale;  
LONG tim_addr, old_addr;
```

```
vex_timv(handle, tim_addr, &old_addr, &scale);
```

Parameter Block Binding

Control	Input	Output
control(0) = 118		intout(0) = scale
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 1		
control(5) = 0		
control(6) = handle		
control(7-8) = tim_addr		
control(9-10) = old_addr		

V_SHOW_C (7AH)

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

`v_show_c` and `v_hide_c` (7BH) are closely related. Once the cursor is visible, a single `v_hide_c` call removes the cursor from the display surface. The VDI keeps track of the number of times `v_hide_c` is called. `v_show_c` must be called the same number of times for the cursor to reappear. For example, if `v_hide_c` is called four times, `v_show_c` must be called four times for the cursor to appear.

`v_show_c` does, however, provide a reset flag. If the calling routine sets the flag is zero, the cursor appears on the screen regardless of the number of `v_hide_c` calls. A nonzero value for the reset flag affects `v_show_c` as described above.

Input Arguments

<code>handle</code>	Device handle
<code>reset</code>	Reset flag:
	zero Ignore number of <code>v_hide_c</code> calls
	nonzero Normal <code>v_show_c</code> functionality

Sample Call to C Language Binding

```
WORD v_show_c();  
WORD handle, reset;  
  
v_show_c(handle, reset);
```

Parameter Block Binding

Control	Input	Output
---------	-------	--------

control(0) = 122

intin(0) = reset

control(1) = 0

control(2) = 0

control(3) = 1

control(4) = 0

control(5) = 0

control(6) = handle

V_HIDE_C (7BH)

v_hide_c removes the cursor from the display surface. This state is the default condition set when the device is opened with v_opnwk (1H) or v_opnwk (64H). The cursor can appear in a new position when the application calls v_show_c (7AH) because the VDI updates the cursor position based on information input from a mouse.

See the description of v_show_c on page 7-24 for a discussion of how the number of v_hide_c calls affects the v_show_c function.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_hide_c();  
WORD  handle;  
  
v_hide_c(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 123		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		

VQ_MOUSE (7CH)

vq_mouse returns the current state of the mouse buttons. The state of the leftmost mouse button is returned in the Least Significant Bit of status. A bit value of 0 indicates the button is up; a bit value of 1 indicates the button is currently being pressed.

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

Input Arguments

handle Device handle

Output Arguments

status Mouse button state:

- 0 Button up
- 1 Button pressed

px x position of cursor in NDC/RC units

py y position of cursor in NDC/RC units

Sample Call to C Language Binding

```
WORD vq_mouse();
WORD handle, status, px, py;

vq_mouse(handle, &status, &px, &py);
```

Parameter Block Binding

Control	Input	Output
control(0) = 124		intout(0) = status
control(1) = 0		ptsout(0) = px
control(2) = 1		ptsout(1) = py
control(3) = 0		
control(4) = 1		
control(5) = 0		
control(6) = handle		

VEX_BUTV (7DH)

With `vex_butv`, the calling routine can designate the address of application-dependent code that is to receive control each time the state of the mouse buttons changes. This function passes control to the application-dependent code after the button state is decoded, but before the driver button state changes.

Input to `vex_butv` is a two-word pointer, `usercode`. This argument indicates the starting address of the code to receive control when the mouse button state changes. `vex_butv` returns a two-word pointer, `savecode`, 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 instruction; a processor-dependent register contains the mouse button keys. Keys are encoded by the same rules that apply to the `vq_mouse` (7CH) function. When complete, the application-dependent code should perform 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 of altering the buttons before they are used by the driver.

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

Note: It is the responsibility of the application-dependent code to save and restore any registers used. When the application-dependent code is invoked, interrupts are disabled. The application should not enable interrupts.

This function is available only to the application that opened the physical workstation. The device handle must be the root or physical device handle.

Input Arguments

<code>handle</code>	Physical device handle
<code>usercode</code>	Address of application mouse button state change routine

Output Arguments

savecode Address of old mouse button state change routine

Sample Call to C Language Binding

```
WORD vex_butv();  
WORD handle;  
LONG usercode, savecode;  
  
vex_butv(handle, usercode, &savecode);
```

Parameter Block Binding

Control	Input	Output
control(0) = 125		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		
control(7-8) = usercode		
control(9-10) = savecode		

VEX_MOTV (7EH)

`vex_motv` allows the calling routine to designate the address of application-dependent code that is to receive control each time the mouse moves to a new location. The application-dependent code 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.

Input to this function is a two-word pointer, `usercode`. This argument indicates the starting address of the code to receive control when the mouse moves. `vex_motv` returns a two-word pointer, `savecode`, to the address of the old mouse movement routine.

When the mouse moves, the application-dependent code is invoked via a processor-dependent instruction. The new `x` and `y` locations are contained in processor-dependent registers. Upon completion, the application-dependent code performs a processor-dependent return instruction with the `x,y` mouse position the driver is to store in the appropriate hardware registers. This procedure allows the `x,y` position to be altered before it is used by the driver. See Appendix E for processor-specific instructions and register names.

Note: 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.

This function is available only to the application that opened the physical workstation. The device handle must be the "root" or physical device handle.

Input Arguments

<code>handle</code>	Physical device handle
<code>usercode</code>	Address of application mouse movement routine

Output Arguments

savecode Address of the old mouse movement routine

Sample Call to C Language Binding

```
WORD vex_motv();
WORD handle;
LONG usercode, savecode;

vex_motv(handle, usercode, &savecode);
```

Parameter Block Binding

Control	Input	Output
control(0) = 126		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		
control(7-8) = usercode		
control(9-10) = savecode		

VEX_CURV (7FH)

`vex_curv` allows the calling routine to designate the address of application-dependent code that is to receive control each time the cursor is drawn. The application can completely take over drawing the cursor or perform some action and then have the VDI draw the cursor. Control is passed to the application whenever the cursor position should be updated.

Input to this function is a two-word pointer, `usercode`. This argument indicates the starting address of the code to receive control when a cursor is drawn. `vex_curv` returns a pointer, `savecode`, to the address of the old cursor draw routine.

The application-dependent code is invoked with a processor-dependent 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 `psavcode`. This will cause the VDI to draw a cursor. When it is done, the application should perform a processor-dependent return instruction. See Appendix E for processor-specific instructions and register names.

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

This function is available only to the application that opened the physical workstation. The device handle must be the "root" or physical device handle.

Input Arguments

<code>handle</code>	Physical device handle
<code>usercode</code>	Address of application cursor draw routine

Output Arguments

savecode Address of the old cursor draw routine

Sample Call to C Language Binding

```
WORD vex_curv();
WORD handle;
LONG usercode, savecode;

vex_curv(handle, usercode, &savecode);
```

Parameter Block Binding

Control	Input	Output
control(0) = 127		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 0		
control(6) = handle		
control(7-8) = usercode		
control(9-10) = savecode		

VQ_KEY_S (80H)

vq_key_s returns the current state of the keyboard's Control, Shift, and Alt keys as a bit-encoded value in status. The bit assignments are:

- 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 that the key is being pressed.

Input Arguments

handle Device handle

Output Arguments

status Keyboard state

Sample Call to C Language Binding

```
WORD  vq_key_s();
WORD  handle, status;
```

```
vq_key_s(handle, &status);
```

Parameter Block Binding

Control	Input	Output
control(0) = 128		intout(0) = status
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 1		
control(5) = 0		
control(6) = handle		

End of Section 7

Inquire Functions

The inquire functions return the current settings for device-specific attributes such as color index values, polyline types, and fill area styles.

Table 8-1. VDI Inquire Functions

Function	Page	Purpose
vq_color (1AH)	8-2	Inquire color representation
vq_cellarray (1BH)	8-4	Inquire cell array
vql_attributes (23H)	8-6	Inquire current polyline attributes
vqm_attributes (24H)	8-8	Inquire current polymarker attributes
vqf_attributes (25H)	8-10	Inquire current fill area attributes
vqt_attributes (26H)	8-12	Inquire current graphic text attributes
vq_extnd (66H)	8-14	Return extended device-specific information
vqin_mode (73H)	8-18	Inquire input mode
vqt_extent (74H)	8-19	Inquire text extent
vqt_width (75H)	8-22	Inquire character cell width
vqt_name (82H)	8-24	Inquire font name and index
vqt_font_info (83H)	8-26	Inquire current font information
vqt_justified (84H)	8-29	Inquire justified graphics text

VQ_COLOR (1AH)

Inquire Color Representation returns the units of RGB intensity associated with a color index. You can specify that `vq_color` is to return either the units of RGB intensity as set by a call to `vs_color` (EH) or the actual RGB values used by the device driver (these are the realized values selected by the driver if a color is not available). `vs_color` is described in Section 5, "Attribute Functions."

The Parameter Block (assembly language) binding for `vq_color` returns -1 in `intout(0)` if the specified index is out of range. The C binding returns only the units of RGB intensity.

Input Arguments

<code>handle</code>	Device handle
<code>index</code>	Requested color index
<code>set_flag</code>	Set or realized flag: 0 set (return RGB intensities for requested index) 1 realized (return color values realized on device)

Output Arguments

<code>rgb[0]</code>	Red intensity (in tenths of percent 0-1000)
<code>rgb[1]</code>	Green intensity
<code>rgb[2]</code>	Blue intensity

Sample Call to C Language Binding

```
WORD vq_color();  
WORD handle, index, set_flag, rgb[3];  
  
vq_color(handle, index, set_flag, rgb);
```

Parameter Block Binding

Control	Input	Output
contrl(0) = 26	intin(0) = index	intout(0) = -1 if index invalid
contrl(1) = 0	intin(1) = set_flag	intout(1) = rgb[0]
contrl(2) = 0		intout(2) = rgb[1]
contrl(3) = 2		intout(3) = rgb[2]
contrl(4) = 4		
contrl(6) = Device handle		

VQ_CELLARRAY (1BH)

vq_cellarray 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 defined by the pxyarray arguments. vq_cellarray returns -1 to indicate that it could not determine a color index for a particular pixel.

All input coordinates are specified according to the current coordinate system (NDC/RC).

Note: This function is not required and may not be supported by all devices drivers.

Input Arguments

handle	Device handle
pxyarray[0]	x-coordinate of rectangle's lower left corner
pxyarray[1]	y-coordinate of rectangle's lower left corner
pxyarray[2]	x-coordinate of rectangle's upper right corner
pxyarray[3]	y-coordinate of rectangle's upper right corner
row_length	Length of each row in color index array
num_rows	Number of rows in color index array

Output Arguments

el_used	Number of elements used in each row (number of columns) of color index array
rows_used	Number of rows used in color index array
status	Invalid value flag:
	0 No errors
	1 If a color value could not be determined for some pixel

colarray[0] Color index array, stored by row
 .
 .
 colarray[n] Last row of color index

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 27	ptsin(0) = pxyarray[0]	intout(0) = colarray[0]
control(1) = 2	ptsin(1) = pxyarray[1]	.
control(2) = 0	ptsin(2) = pxyarray[2]	.
control(3) = 0	ptsin(3) = pxyarray[3]	intout(0) = colarray[n]
control(4) = n		
control(5) = 0		
control(6) = handle		
control(7) = row_length		
control(8) = num_rows		
control(9) = el_used		
control(10) = rows_used		
control(11) = status		

VQL_ATTRIBUTES (23H)

vql_attributes returns the current setting of the attributes that affect polylines.

Input Arguments

handle Device handle

Output Arguments

attrib[0] Current polyline line type:

- 1 Solid
- 2 Long dash
- 3 Dot
- 4 Dash,dot
- 5 Dash
- 6 Dash,dot,dot
- 7 User-defined; see vsi_udsty (71H) on page 5-10
- 8-n Device-dependent

See vsi_type (FH) in Section 5 for more information.

attrib[1] Current polyline line color index

attrib[2] Current writing mode:

- 1 Replace mode
- 2 Transparent mode
- 3 XOR mode
- 4 Reverse transparent mode

See vswr_mode (20H) in Section 5 for more information.

- attrib[3]** End style for beginning point of polyline (not returned by C binding):
- 0 Squared
 - 1 Arrow
 - 2 Rounded
- attrib[4]** End style for ending point of polyline (not returned by C binding)
- attrib[5]** Current line width, in x-axis NDC/RC units

Note that only the Parameter Block (assembly language) binding for `vql_attributes` returns the polyline end styles. The C binding returns the line type in `attrib[0]`, the line color in `attrib[1]`, the current writing mode in `attrib[2]`, and the current line width in `attrib[3]`.

Sample Call to C Language Binding

```
WORD vql_attributes();
WORD handle, attrib[4];
```

```
vql_attributes(handle, attrib);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 35</code>		<code>intout(0) = attrib[0]</code>
<code>control(1) = 0</code>		<code>intout(1) = attrib[1]</code>
<code>control(2) = 1</code>		<code>intout(2) = attrib[2]</code>
<code>control(3) = 0</code>		<code>intout(3) = attrib[3]</code>
<code>control(4) = 5</code>		<code>intout(4) = attrib[4]</code>
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		<code>pstout(0) = attrib[5]</code>
		<code>ptsout(1) = 0</code>

VQM_ATTRIBUTES (24H)

vqm_attributes returns the current setting of the attributes that affect polymarkers.

Input Arguments

handle Device handle

Output Arguments

attrib[0] Current marker type:

- 1 Dot
- 2 Plus
- 3 Asterisk
- 4 Square
- 5 Diagonal Cross
- 6 Diamond
- 7-n User-defined

attrib[1] Current polymarker color index

attrib[2] Current writing mode

- 1 Replace mode
- 2 Transparent mode
- 3 XOR mode
- 4 Reverse transparent mode

See vswr_mode (20H) in Section 5 for more information.

attrib[3] Current polymarker width, in x-axis NDC/RC units

attrib[4] Current polymarker height, in y-axis NDC/RC units

Note that only the Parameter Block (assembly) binding for vqm_attributes returns the current polymarker width. The C binding returns the marker type in attrib[0], the marker color index in attrib[1], the current writing mode in attrib[2], and the marker height in attrib[3].

Sample Call to C Language Binding

```
WORD  vqm_attributes();  
WORD  handle, attrib[4];  
  
vqm_attributes(handle, attrib);
```

Parameter Block Binding

Control	Input	Output
control(0) = 36		intout(0) = attrib[0]
control(1) = 0		intout(1) = attrib[1]
control(2) = 1		intout(2) = attrib[2]
control(3) = 0		
control(4) = 3		ptsout(0) = attrib[3]
control(5) = 0		ptsout(1) = attrib[4]
control(6) = handle		

VQF_ATTRIBUTES (25H)

vqf_attributes returns the current setting of the attributes that affect fill areas.

Input Arguments

handle Device handle

Output Arguments

attrib[0] Current fill area interior style:

- 0 Hollow
- 1 Solid
- 2 Pattern
- 3 Hatch
- 4 User-defined

See vsf_interior (17H) in Section 5 for more information.

attrib[1] Current fill area color index

attrib[2] Current fill area style index, refer to vsf_style (18H) on 5-35

attrib[3] Current writing mode:

- 1 Replace mode
- 2 Transparent mode
- 3 XOR mode
- 4 Reverse transparent mode

See vswr_mode (20H) in Section 5 for more information.

attrib[4] Current fill perimeter status:

zero Invisible
nonzero Visible

Note that only the Parameter Block (assembly) binding for `aqf_attributes` returns the current fill perimeter status. The C binding does not return `attrib[4]`.

Sample Call to C Language Binding

```
WORD vqf_attributes();  
WORD handle, attrib[4];  
  
vqf_attributes(handle, attrib);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 37</code>		<code>intout(0) = attrib[0]</code>
<code>control(1) = 0</code>		<code>intout(1) = attrib[1]</code>
<code>control(2) = 0</code>		<code>intout(2) = attrib[2]</code>
<code>control(3) = 0</code>		<code>intout(3) = attrib[3]</code>
<code>control(4) = 5</code>		<code>intout(4) = attrib[4]</code>
<code>control(5) = 0</code>		
<code>control(6) = handle</code>		

VQT_ATTRIBUTES (26H)

`vqt_attributes` returns the current setting of all attributes that affect graphic text. These attributes include text size, text color, and cell dimensions.

Input Arguments

handle Device handle

Output Arguments

attrib[0] Current graphic text font, see Table 5-6 on 5-25

attrib[1] Current graphic text color index

attrib[2] Current angle of rotation of text baseline (in tenths of degrees 0-3600)

attrib[3] Current horizontal alignment:

- 0 Left justified
- 1 Center justified
- 2 Right justified

See `vst_alignment` (27H) in Section 5 for more information.

attrib[4] Current vertical alignment:

- 0 Baseline (default)
- 1 Half line
- 2 Ascent line
- 3 Bottom line
- 4 Descent line
- 5 Top line

attrib[5]	Current writing mode:
	1 Replace mode
	2 Transparent mode
	3 XOR mode
	4 Reverse transparent mode
	See <code>vswr_mode</code> (20H) in Section 5 for more information.
attrib[6]	Current character width in x-axis NDC/RC units
attrib[7]	Current character height in y-axis NDC/RC units
attrib[8]	Current character cell width in x-axis NDC/RC units
attrib[9]	Current character cell height in y-axis NDC/RC units

Sample Call to C Language Binding

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

vqt_attributes(handle, attrib);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 38</code>		<code>intout(0) = attrib[0]</code>
<code>control(1) = 0</code>		.
<code>control(2) = 2</code>		.
<code>control(3) = 0</code>		<code>intout(5) = attrib[5]</code>
<code>control(4) = 6</code>		.
<code>control(5) = 0</code>		<code>ptsout(0) = attrib[6]</code>
<code>control(6) = handle</code>		.
		.
		<code>ptsout(3) = attrib[9]</code>

VQ_EXTND (66H)

vq_extnd returns an extended set of device-specific information not included in the Open Workstation, v_opnwk (1H), function. The information type flag allows the calling routine to specify if vq_extnd is to return the v_opnwk output arguments or its extended set of device-specific information. Refer to 3-2 in Section 3, "Control Functions," for a description of the values returned by v_opnwk.

Note that although some values are undefined for the extended set of device-specific information, vq_extnd always returns six output vertices and 45 output integers (57 work_out arguments).

Input Arguments

handle	Device handle
owflag	Information type flag:
	0 Return Open Workstation, v_opnwk (1H) information
	1 Return extended inquire information

Output Arguments

work_out[0]	Type of screen:
	0 not screen
	1 separate alpha and graphic controllers and separate screens
	2 separate alpha and graphic controllers with a common screen
	3 common alpha and graphic controller with separate image memory
	4 common alpha and graphic controller with common image memory

- work_out[1]** Number of background colors in color palette
- On some devices, this may be different from the number of colors returned by `v_opnwk (1H)` in `work_out[39]`.
- work_out[2]** Supported text effects, bit pattern, see `vst_effects (6AH)` in Section 5
- work_out[3]** Scale rasters:
- 0 Scaling not possible
 - 1 Scaling possible
- work_out[4]** Number of planes for raster operations
- work_out[5]** Color lookup table support:
- 0 Table not supported
 - 1 Table supported
- work_out[6]** Performance factor, number of 16 x 16 pixel raster operations per second
- work_out[7]** Contour Fill, `v_contourfill (67H)`, flag:
- 0 Unavailable
 - 1 Available
- work_out[8]** Character rotation support:
- 0 Not supported
 - 1 90-degree increments only
 - 2 Arbitrary angles
- work_out[9]** Number of writing modes available
- work_out[10]** Highest level of input mode available:
- 0 None
 - 1 Request
 - 2 Sample

work_out[11] Text alignment, *vst_alignment* (27H), flag:

- 0 Unavailable
- 1 Available

work_out[12] Inking capability flag:

- 0 Device cannot ink
- 1 Device can ink

work_out[13] Rubberbanding capability flag:

- 0 Device incapable
- 1 Capable of rubberband lines
- 2 Capable of both rubberband lines and rectangles

work_out[14] Maximum number of vertices for *v_pline* (6H), *v_pmarker* (7H), or *v_fillarea* (9H):

- 1 No maximum

work_out[15] Maximum length of *intin* array:

- 1 No maximum

work_out[16] Number of buttons available on the mouse

work_out[17] Styles supported for wide lines:

- 0 No
- 1 Yes

work_out[18] Writing modes for wide lines:

- 0 No
- 1 Yes

work_out[19] Clipping, *vs_clip* (81H), flag:

- 0 Clipping disabled
- 1 Clipping enabled

work_out[20]
to
work_out[44] Reserved, contain zeros

work_out[45] Upper left x-coordinate of the clipping rectangle in
x-axis NDC/RC units

work_out[46] Upper left y-coordinate of the clipping rectangle in
y-axis NDC/RC units

work_out[47] Lower right x-coordinate of the clipping rectangle in
x-axis NDC/RC units

work_out[48] Lower right y-coordinate of the clipping rectangle in
y-axis NDC/RC units

work_out[49]
to
work_out[56] Reserved, contain zeros

Sample Call to C Language Binding

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

vq_extnd(handle, owflag, work_out);
```

Parameter Block Binding

Control	Input	Output
control(0) = 102	intin(0) = owflag	intout(0) = work_out[0]
control(1) = 0		.
control(2) = 6		.
control(3) = 1		intout(44) = work_out[44]
control(4) = 45		.
control(5) = 0		pstout(0) = work_out[45]
control(6) = handle		.
		pstout(11) = work_out[56]

VQIN_MODE (73H)

vqin_mode returns the current input mode for the specified logical device: locator, valuator, choice, or string.

Input Arguments

handle	Device handle
dev_type	Logical input device:
	1 Locator
	2 Valuator
	3 Choice
	4 String

Output Arguments

input_mode	Input mode:
	1 Request
	2 Sample

Sample Call to C Language Binding

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

vqin_mode(handle, dev_type, &input_mode);
```

Parameter Block Binding

```
control(0) = 115          intin(0) = dev_type          intout(0) = input_mode
control(1) = 0
control(2) = 0
control(3) = 1
control(4) = 1
control(5) = 0
control(6) = handle
```

VQT_EXTENT (74H)

`vqt_extent` uses the current text attributes to calculate the dimensions of a specified string and returns the coordinates of a rectangle that encloses string. The returned vertex coordinates define an extent rectangle that touches both the x and y axes. The string is in the first quadrant. All text attributes, including style and baseline rotation, affect the calculation.

Output arguments `extent[0]` to `extent[7]` specify string points according to the current coordinate system (NDC or RC).

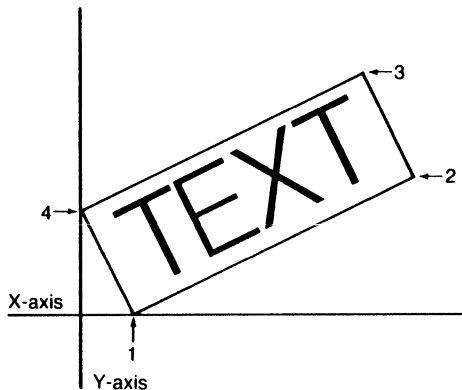


Figure 8-1. Inquire Text Extent Function

Input Arguments

<code>handle</code>	Device handle
<code>string</code>	Character string as ASCII codes

The C binding string is byte-oriented and must be null-terminated. For the Parameter block binding, the string is word-oriented (16-bits). The maximum number of characters equals the size of the `intin` array; see `work_out[15]` of `vq_extnd` (66H).

Output Arguments

extent[0] delta-x for point 1 of string in x-axis NDC/RC units
 extent[1] delta-y for point 1 of string in y-axis NDC/RC units
 extent[2] delta-x for point 2 of string in x-axis NDC/RC units
 extent[3] delta-y for point 2 of string in y-axis NDC/RC units
 extent[4] delta-x for point 3 of string in x-axis NDC/RC units
 extent[5] delta-y for point 3 of string in y-axis NDC/RC units
 extent[6] delta-x for point 4 of string in x-axis NDC/RC units
 extent[7] delta-y for point 4 of string in y-axis NDC/RC units

Sample Call to C Language Binding

```

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

```

vqt_extent(handle, string, extent);
  
```

Parameter Block Binding

Control	Input	Output
control(0) = 116	intin = string	ptsout(0) = extent[0]
control(1) = 0		.
control(2) = 4		.
control(3) = number of chars in string		ptsout(7) = extent[7]
control(4) = 0		
control(5) = 0		
control(6) = handle		

Parameter Block Binding

Control	Input	Output
control(0) = 116	intin = string	ptsout(0) = extent[0]
control(1) = 0		.
control(2) = 4		.
control(3) = number of chars in string		ptsout(7) = extent[7]
control(4) = 0		.
control(5) = 0		.
control(6) = handle		.

VQT_WIDTH (75H)

This function returns the character cell width for a specified character in the current font. 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; see Figure 8-2. Special effects and rotation do not apply.

`vqt_width` returns all values in the current coordinate system.

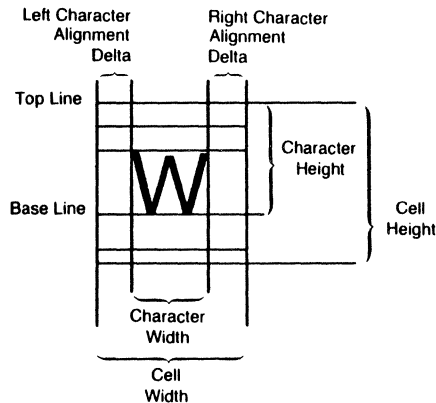


Figure 8-2. Character Cell Definition

Input Arguments

handle	Device handle
character	ASCII Decimal Equivalent (ADE) value of the character in current font

Output Arguments

status	ADE value of the specified character; -1 if invalid character
cell_width	Cell width of the character in x-axis NDC/RC units
left_delta	Left character alignment delta in x-axis NDC/RC units
right_delta	Right character alignment delta in y-axis NDC/RC units

Sample Call to C Language Binding

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

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

Parameter Block Binding

Control	Input	Output
control(0) = 117	intin(0) = character	intout(0) = status
control(1) = 0		
control(2) = 3		ptsout(0) = cell_width
control(3) = 1		ptsout(1) = 0
control(4) = 1		ptsout(2) = left_delta
control(5) = 0		ptsout(3) = 0
control(6) = handle		ptsout(4) = right_delta
		ptsout(5) = 0

VQT_NAME (82H)

vqt_name returns a 32-character ASCII Decimal Equivalent (ADE) string that describes a specified font. The calling routine specifies a font by its element number (1 to the number of fonts available).

The first 16 characters of the ADE string name the font. The next 16 characters describe the font's style and weight. The string is null terminated. See Table 8-2 for a sample of the possible configurations.

vqt_name also returns the font index number. This number is used to access the font with vst_font (15H).

Table 8-2. Face Names and Styles

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

Input Arguments

handle Device handle

element_num Element number

Output Arguments

index Font ID number

name[0]
to
name[31] 32-character ADE string

For the Parameter Block (assembly language) binding, the string is word-oriented. For the C binding, the string is byte-oriented.

Sample Call to C Language Binding

```
WORD  vqt_name();
WORD  index, handle, element_num;
BYTE  name[32];
```

```
index = vqt_name(handle, element_num, name);
```

Parameter Block Binding

Control	Input	Output
control(0) = 130	intin(0) = element_num	intout(0) = index
control(1) = 0		intout(1) = name[0]
control(2) = 0		.
control(3) = 1		.
control(4) = 33		intout(32) = name[31]
control(5) = 0		
control(6) = handle		

VQT_FONT_INFO (83H)

This function returns size information for the current font with the current size and text effects.

Because the text effects can change the cell width and extent, `vqt_font_info` returns a value (`maxwidth`) for the maximum cell width that excludes any text special effects. This allows the use of the width information returned by `vqt_width` (75H).

When the character is skewed, the cell contains left and right offsets as shown in Figure 8-3.

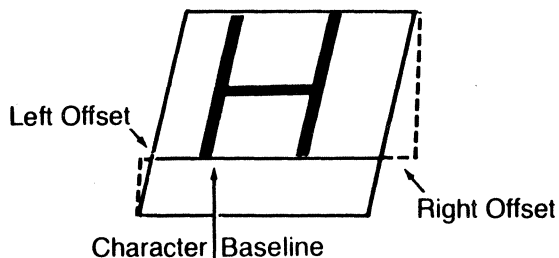


Figure 8-3. Skewed Character Offsets

Input Arguments

handle Device handle

Output Arguments

minADE Minimum ADE (ASCII Decimal Equivalent); this is the first character in the font

maxADE Maximum ADE, the last character in the font

maxwidth	Maximum cell width not including special effects
distances[0]	Bottom line distance relative to baseline
distances[1]	Descent line distance relative to baseline in y-axis NDC/RC units
distances[2]	Half line distance relative to baseline in y-axis NDC/RC units
distances[3]	Ascent line distance relative to baseline in y-axis NDC/RC units
distances[4]	Top line distance relative to baseline in y-axis NDC/RC units
effects[0]	Special effects delta x, the current special effects increase character width by this amount in x-axis NDC/RC units
effects[1]	Left offset, positive value relative to position in x-axis NDC/RC units (see Figure 8-3)
effects[2]	Right offset in x-axis NDC/RC units (see Figure 8-3)

Sample Call to C Language Binding

```
WORD vqt_font_info();  
WORD handle, minADE, maxADE, distances[5], maxwidth, effects[3];  
  
vqt_font_info(handle, &minADE, &maxADE, distances, &maxwidth,  
effects);
```


Parameter Block Binding

Control	Input	Output
control(0) = 131		intout(0) = minADE
control(1) = 0		intout(1) = maxADE
control(2) = 5		
control(3) = 0		ptsout(0) = maxwidth
control(4) = 2		ptsout(1) = distances[0]
control(5) = 0		ptsout(2) = effects[0]
control(6) = handle		ptsout(3) = distances[1]
		ptsout(4) = effects[1]
		ptsout(5) = distances[2]
		ptsout(6) = effects[2]
		ptsout(7) = distances[3]
		ptsout(8) = 0
		ptsout(9) = distances[4]

VQT_JUSTIFIED (84H)

Inquire Justified Graphics Text returns the x- and y-axis offsets for each character in a string given a text alignment point and physical string length.

See `v_justified` (B-AH) on page 4-27 in Section 4 for related information.

Input Arguments

<code>handle</code>	Device handle
<code>word_space</code>	Word spacing flag: zero Do not modify inter-word spacing nonzero Modify inter-word spacing
<code>char_space</code>	Character spacing flag: zero Do not modify inter-character spacing nonzero Modify inter-character spacing
<code>string</code>	ASCII character string
<code>x</code>	x-coordinate of the text alignment point
<code>y</code>	y-coordinate of the text alignment point
<code>length</code>	Requested length of the string, in x-axis units

Output Arguments

offsets[0]	x-axis offset to base-left of the first character in string
offsets[1]	y-axis offset to base-left of the first character in string
	.
	.
	.
offsets[2*n-2]	x-axis offset to base-left of the last character in the string ("n" is used here for the number of characters in the string)
offsets[2*n-1]	y-axis offset to base-left of the last character in the string ("n" is used here for the number of characters in the string)

Sample Call to C Language Binding

```
WORD vqt_justified();  
WORD handle, x, y, length, word_space, char_space, string[],  
offsets[];  
  
vqt_justified(handle, x, y, string, length, word_space, char_space,  
offsets);
```

Parameter Block Binding

Control	Input	Output
control(0) = 132	intin(0) = word_space	ptsout(0) = offsets[0]
control(1) = 2	intin(1) = char_space	ptsout(1) = offsets[1]
control(2) = n	intin(2) = string[0]	.
control(3) = 2 + n	.	.
control(4) = 0	.	ptsout(j) = offsets[j]
control(5) = 0	intin(2+n) = string[2+n]	
control(6) = handle		
	ptsin(0) = x	
	ptsin(1) = y	
	ptsin(2) = length	
	ptsin(3) = 0	

Note: "n" is used here as the number of characters in the string.

End of Section 8

Escape Functions

The escape functions access the special capabilities of a graphics device. The VDI predefines some escape functions, such as those for controlling the cursor on an alpha screen. Others, whose opcodes must be greater than 5-64H, can be defined for specific devices.

Table 9-1 lists and briefly describes the VDI escape functions.

Table 9-1. Escape Function Identifiers

Function	Page	Purpose
vq_chcells (5-1H)	9-3	Return the number of alpha cursor addressable rows and columns
v_exit_cur (5-2H)	9-4	Exit alpha mode for graphics mode
v_enter_cur (5-3H)	9-5	Exit graphics mode for alpha mode
v_curup (5-4H)	9-6	Move alpha cursor up one row, maintain column position
v_curdown (5-5H)	9-7	Move alpha cursor down one row, maintain column position
v_currigh (5-6H)	9-8	Move alpha cursor right one column, maintain row position
v_curleft (5-7H)	9-9	Move alpha cursor left one column, maintain row position
v_curhome (5-8H)	9-10	Move alpha cursor to home position
v_eeos (5-9H)	9-11	Erase from alpha cursor position to end of screen
v_eeol (5-AH)	9-12	Erase from alpha cursor to end of line
vs_curaddress (5-BH)	9-13	Move alpha cursor to specified location
v_curtex (5-CH)	9-14	Display text string at alpha cursor location

Table 9-1. (Cont'd)

Function	Page	Description
v_rvon (5-DH)	9-15	Turn on reverse video
v_rvoff (5-EH)	9-16	Turn off reverse video
vq_curaddress (5-FH)	9-17	Return current alpha cursor address
vq_tabstatus (5-10H)	9-18	Inquire tablet status
v_hardcopy (5-11H)	9-19	Copy physical screen to printer
v_dspcur (5-12H)	9-20	Place graphic cursor at specified coordinates
v_rmcur (5-13H)	9-21	Remove last graphic cursor
v_form_adv (5-14H)	9-22	Form advance
v_output_window (5-15H)	9-23	Output window to printer
v_clear_disp_list (5-16H)	9-25	Clear the printer display list
v_bit_image (5-17H)	9-26	Print the specified bit image file
vq_scan (5-18H)	9-29	Return printer scan heights and number of head passes per page
v_alpha_text (5-19H)	9-31	Print specified alpha text string
vs_palette (5-3CH)	9-33	Select color palette
v_sound (5-3DH)	9-34	Generate specified tone
vs_mute (5-3EH)	9-35	Set/Clear tone muting flag
vt_resolution (5-51H)	9-36	Set tablet axis resolution in lines/inch
vt_axis (5-52H)	9-37	Set tablet axis resolution in lines
vt_origin (5-53H)	9-38	Set tablet x and y origin
vq_tdimensions (5-54H)	9-39	Return tablet x and y dimensions
vt_alignment (5-55H)	9-40	Set tablet alignment
vsp_film (5-5BH)	9-41	Set camera film type and exposure time
vqp_filmname (5-5CH)	9-42	Inquire camera film name
vsc_expose (5-5DH)	9-44	Disable or enable film exposure for frame preview
v_meta_extents (5-62H)	9-45	Update extent information in metafile header
v_write_meta (5-63H)	9-46	Write a user-defined metafile item
vm_filename (5-64H)	9-47	Use specified filename to rename metafile

VQ_CHCELLS (5-1H)

vq_chcells returns 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 Arguments

handle Device handle

Output Arguments

rows Number of addressable rows on the screen, (-1 indicates cursor addressing not possible)

columns Number of addressable columns on the screen, (-1 indicates cursor addressing not possible)

Sample Call to C Language Binding

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

vq_chcells(handle, &rows, &columns);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		intout(0) = rows
control(1) = 0		intout(1) = columns
control(2) = 0		
control(3) = 0		
control(4) = 2		
control(5) = 1		
control(6) = handle		

V_EXIT_CUR (5-2H)

This function causes the graphics device to enter graphics mode if graphics mode is different from alpha mode. `v_exit_cur` exits alpha cursor addressing mode explicitly, properly transferring from alpha to graphics mode. `v_exit_cur` also clears the graphics screen.

If the specified device is a metafile, `v_exit_cur` writes a metafile item to the metafile buffer.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_exit_cur();
WORD  handle;

v_exit_cur(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(5) = 2		
control(6) = handle		

V_ENTER_CUR (5-3H)

`v_enter_cur` causes the graphics device to exit graphics mode if graphics mode is different from alpha mode. This function enters alpha cursor addressing mode explicitly, properly transferring from graphics to alpha mode. It also clears the alpha screen.

`v_enter_cur` also returns the cursor to the home position on the display device.

If this function is called by a metafile, `v_enter_cur` writes a metafile item to the metafile buffer.

Input Arguments

handle Device handle

Sample Call to C Language Binding

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

```
v_enter_cur(handle);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>		
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 0</code>		
<code>control(4) = 0</code>		
<code>control(5) = 3</code>		
<code>control(6) = handle</code>		

V_CURUP (5-4H)

This function moves the alpha cursor up one row without altering its horizontal position. If the cursor is already at the top margin, v_curup takes no action.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD v_curup();
```

```
WORD handle;
```

```
v_curup(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 4		
control(6) = handle		

V_CURDOWN (5-5H)

`v_curdown` moves the alpha cursor down one row without altering its horizontal position. No action occurs if the cursor is already at the bottom margin.

Input Arguments

`handle` Device handle

Sample Call to C Language Binding

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

```
v_curdown(handle);
```

Parameter Block Binding

Control	Input	Output
<code>control(0)</code>		<code>5</code>
<code>control(1)</code>		<code>0</code>
<code>control(2)</code>		<code>0</code>
<code>control(3)</code>		<code>0</code>
<code>control(4)</code>		<code>0</code>
<code>control(5)</code>		<code>5</code>
<code>control(6)</code>		<code>handle</code>

V_CURRIGHT (5-6H)

This function moves the alpha cursor right one column without altering its vertical position. If the cursor is already at the right margin, v_currright performs no action.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_currright();  
WORD  handle;
```

```
v_currright(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 6		
control(6) = handle		

V_CURLEFT (5-7H)

v_curleft moves the alpha cursor left one column without altering its vertical position. This function performs no action if the cursor is already at the left margin.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD v_curleft();  
WORD handle;
```

```
v_curleft(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 7		
control(6) = handle		

V_CURHOME (5-8H)

This function moves the alpha cursor to the home position, usually the upper left character cell of the display device.

Input Arguments

handle Device handle

Sample Call to C Language Binding

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

```
v_curhome(handle);
```

Parameter Block Binding

Control	Input	Output
---------	-------	--------

```
control(0) = 5  
control(1) = 0  
control(2) = 0  
control(3) = 0  
control(4) = 0  
control(5) = 8  
control(6) = handle
```

V_EEOS (5-9H)

v_eeos 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 Arguments

handle Device handle

Sample Calling to C Language Binding

```
WORD v_eeos();  
WORD handle;
```

```
v_eeos(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 9		
control(6) = handle		

V_EEOL (5-AH)

This function 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 Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_eeol();  
WORD  handle;
```

```
v_eeol(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 10		
control(6) = handle		

VS_CURADDRESS (5-BH)

`vs_curaddress` moves the alpha cursor directly to the row and column specified. The calling routine can specify any location on the display surface. `vs_curaddress` sets addresses beyond the displayable range of the screen to the nearest value that is within the displayable range of the screen.

Use `vq_chcells` (5-1H), described on page 9-3, to determine the number of addressable rows and columns.

Input Arguments

<code>handle</code>	Device handle
<code>row</code>	Row number (1 to maximum number of rows)
<code>column</code>	Column number (1 to maximum number of columns)

Sample Call to C Language Binding

```
WORD vs_curaddress();
WORD handle, row, column;

vs_curaddress(handle, row, column);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = row</code>	
<code>control(1) = 0</code>	<code>intin(1) = column</code>	
<code>control(2) = 0</code>		
<code>control(3) = 2</code>		
<code>control(4) = 0</code>		
<code>control(5) = 11</code>		
<code>control(6) = handle</code>		

V_CURTEXT (5-CH)

`v_curtext` displays a specified string of alpha text starting at the current position of the alpha cursor. It moves the cursor right one column for each character in the string. The alpha text attribute currently in effect determines how the text is displayed (see `v_rvon` (5-DH) and `v_revoff` on pages 9-15 and 9-16, respectively).

The device must be in alpha cursor addressing mode before using `v_curtext`, see `v_enter_cur` (5-3H) on page 9-5.

Input Arguments

handle Device handle

string ASCII Decimal Equivalent (ADE) text string

The string is word-oriented for the Parameter Block binding. For the C binding, the string is byte-oriented.

Sample Call to C Lanugage Binding

WORD `v_curtext();`

WORD `handle;`

BYTE `*string;`

`v_curtext(handle, string);`

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin = string</code>	
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = n</code>		
<code>control(4) = 0</code>		
<code>control(5) = 12</code>		
<code>control(6) = handle</code>		

V_RVON (5-DH)

This function causes all subsequent alpha text to be displayed in reverse video.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  r_von();  
WORD  handle;
```

```
v_rvon(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 13		
control(6) = handle		

V_RVOFF (5-EH)

v_revoff displays all subsequent alpha text in normal video format.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_revoff();  
WORD  handle;  
  
v_revoff(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 14		
control(6) = handle		

VQ_CURADDRESS (5-FH)

vq_curaddress returns the current position of the alpha cursor in row, column coordinates.

Input Arguments

handle Device handle

Output Arguments

row Row number (1 to the maximum number of rows)

column Column number (1 to the maximum number of columns)

Sample Call to C Language Binding

```
WORD vq_curaddress();
WORD handle, row, handle;
```

```
vq_curaddress(handle, &row, &column);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		intout(0) = row
control(1) = 0		intout(1) = column
control(2) = 0		
control(3) = 0		
control(4) = 2		
control(5) = 15		
control(6) = handle		

VQ_TABSTATUS (5-10H)

vq_tabstatus returns the availability status of a graphics tablet, mouse, joystick, or other similar device.

Input Arguments

handle Device handle

Output Arguments

status Tablet status:
 0 Tablet not available
 1 Tablet available

Sample Call to C Language Binding

```
WORD vq_tabstatus();
WORD status, handle;
```

```
status = vq_tabstatus(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		intout(0) = status
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 1		
control(5) = 16		
control(6) = handle		

V_HARDCOPY (5-11H)

`v_hardcopy` causes the specified device to generate a hard copy. This function is device-specific; it copies the physical screen to a printer or other attached hard copy device.

`v_hardcopy` is not required and might not be supported by all device drivers (see "Required Functions" in Section 2).

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD v_hardcopy();
WORD handle;

v_hardcopy(handle);
```

Parameter Block Binding

Control	Input	Output
<code>control(0)</code>		
<code>control(1)</code>		
<code>control(2)</code>		
<code>control(3)</code>		
<code>control(4)</code>		
<code>control(5)</code>		
<code>control(6)</code>	handle	

V_DSPCUR (5-12H)

This function places a graphic cursor at the location specified by the calling routine in the current coordinate system.

vsc_form (6FH) defines the type of cursor to be displayed. If sample mode input is supported, the calling routine can use this function to generate the cursor for vsm_locator (1CH), Input Locator-Sample Mode. vsc_form and vsm_locator are described in Section 7, "Input Functions."

v_dspcur applies only to devices capable of locator input (a mouse, joystick, or trackball, for example).

Input Arguments

handle	Device handle
x	x-coordinate of location to place cursor
y	y-coordinate of location to place cursor

Sample Call to C Language Binding

```
WORD v_dspcur();
WORD handle, x, y;

v_dspcur(handle, x, y);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	ptsin(0) = x	
control(1) = 1	ptsin(1) = y	
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 18		
control(6) = handle		

V_RMCUR (5-13H)

v_rmcure removes the last graphic cursor placed on the screen.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_rmcure();  
WORD  handle;  
  
v_rmcure(handle);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		
control(1) = 0		
control(2) = 0		
control(3) = 0		
control(4) = 0		
control(5) = 19		
control(6) = handle		

V_FORM_ADV (5-14H)

v_form_adv advances the printer page. The VDI requires printer driver support of this function.

Use v_form_adv instead of v_clrwk (3H) when you want issue a form feed and retain the current printer display list.

If the specified device is a metafile, v_form_adv writes a metafile item to the metafile buffer.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD  v_form_adv();  
WORD  handle;
```

```
v_form_adv(handle);
```

Parameter Block Binding

Control	Input
---------	-------

```
control(0) = 5  
control(1) = 0  
control(2) = 0  
control(3) = 0  
control(4) = 0  
control(5) = 20  
control(6) = handle
```

V_OUTPUT_WINDOW (5-15H)

This function outputs a rectangular window of a picture to the printer. `v_output_window` is similar to `v_updwk` (4H), except that the calling routine specifies the coordinates (NDC or RC) of a rectangular window area.

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.

The VDI requires that printer drivers support `v_output_window`. If the specified device is a metafile, `v_output_window` writes a metafile item to the metafile buffer.

Input Arguments

<code>handle</code>	Device handle
<code>xyarray[0]</code>	x-coordinate of window corner
<code>xyarray[1]</code>	y-coordinate of window corner
<code>xyarray[2]</code>	x-coordinate of corner diagonal to <code>xyarray[0]</code>
<code>xyarray[3]</code>	y-coordinate of corner diagonal to <code>xyarray[1]</code>

Sample Calling to C Language Binding

```
WORD v_output_window();  
WORD handle, xyarray[4];
```

```
v_output_window(handle, xyarray);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	ptsin(0) = xyarray[0]	
control(1) = 2	ptsin(1) = xyarray[1]	
control(2) = 0	ptsin(2) = xyarray[2]	
control(3) = 0	ptsin(3) = xyarray[3]	
control(4) = 0		
control(5) = 21		
control(6) = handle		

V_CLEAR_DISP_LIST (5-16H)

v_clear_disp_list is required only for printers. It is called to clear the printer display list. This function is similar to v_clrwk (3H) but it does not cause the printer to perform a form advance.

If the device specified by the calling routine is a metafile, v_clear_disp_list writes a metafile item to the metafile buffer.

Input Arguments

handle Device handle

Sample Call to C Language Binding

```
WORD v_clear_disp_list();
```

```
WORD handle;
```

```
v_clear_disp_list(handle);
```

Parameter Block Binding

Control	Input
---------	-------

```
control(0) = 5  
control(1) = 0  
control(2) = 0  
control(3) = 0  
control(4) = 0  
control(5) = 22  
control(6) = handle
```

V_BIT_IMAGE (5-17H)

`v_bit_image` processes a bit image file (see Appendix G, "Bit Image File Format"). Input arguments include a filename and information on image transformation and page placement.

The calling routine provides three parameters to control image transformation:

- pixel aspect ratio
- x-axis scaling
- y-axis scaling

Pixel Aspect Ratio

The aspect flag allows the calling routine to specify whether the pixel aspect ratio defined in the bit image file is to be preserved or ignored. 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. If the pixel aspect ratio is ignored, the printed object will not necessarily have the same aspect ratio it had on the original device.

Axis Scaling

The two axis scaling flags can be set independently of each other. The scaling 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 a rectangle that the calling routine defines in the `xyarray` arguments.

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 caller can use the `h_align` and `v_align` alignment parameters to locate the bit image within the rectangle. These parameters allow any combination of three vertical and three horizontal positions.

Note: The scaled bit image always resides within the scaling rectangle. If a combination of preserved pixel aspect ratio, axis scaling, or alignment causes the scaled bit image to extend beyond an edge of the scaling rectangle, `v_bit_image` clips the bit image to that edge.

If the device specified by the calling routine is a metafile, `v_bit_image` writes a metafile item to the metafile buffer.

Input Arguments

handle	Device handle
filename	Filename for bit image file. For the C language binding, this argument is byte-oriented and must be null-terminated. For the Parameter Block (assembly language) binding, the filename is word-oriented.
aspect	Aspect ratio flag: 0 ignore aspect ratio 1 Honor pixel aspect ratio
x_scale	Scaling for x-axis: 0 Fractional scaling 1 Integer scaling
y_scale	Scaling for y-axis: 0 Fractional scaling 1 Integer scaling
h_align	Horizontal alignment: 0 Left 1 Center 2 Right

v_align	Vertical alignment:
	0 Top
	1 Middle
	2 Bottom
xyarray[0]	Upper left x (if specified)
xyarray[1]	Upper left y (if specified)
xyarray[2]	Lower right x (if specified)
xyarray[3]	Lower right y (if specified)

Sample Call to C Language Binding

```
WORD v_bit_image();
BYTE *filename;
WORD handle, aspect, x_scale, y_scale, h_align, v_align, xyarray[];
```

```
v_bit_image(handle, filename, aspect, x_scale, y_scale, h_align,
            v_align, xyarray);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = aspect	
control(1) = 2	intin(1) = x_scale	
control(2) = 0	intin(2) = y_scale	
control(3) = filename + 5	intin(3) = h_align	
control(4) = 0	intin(4) = v_align	
control(5) = 23	intin(5)	
control(6) = handle	.	
	.	
	intin(n+4) = filename	
	ptsin(0) = xyarray[0]	
	ptsin(1) = xyarray[1]	
	ptsin(2) = xyarray[2]	
	ptsin(3) = xyarray[3]	

VQ_SCAN (5-18H)

vq_scan returns the height, in pixels, of a printer head pass and the number of head passes in a full page for both graphics and alpha modes.

For devices in which the height of a single head pass is not a discrete number of pixels, this function returns a division factor. For example, if the device outputs 12.5 pixels in a single alpha pass and 4 pixels in a single graphics pass, vq_scan returns an alpha scan height of 25, a graphics scan height of 8, and a division factor of 2.

Input Arguments

handle Device handle

Output Arguments

g_height Graphics scan height in scaled pixels
g_slices Number of graphics head passes per page
a_height Alpha scan height in scaled pixels
a_slices Number of alpha head passes per page
factor Scan height division factor

Sample Call to C Language Binding

```
WORD vq_scan();  
WORD handle, g_height, g_slices, a_height, a_slices, factor;  
vq_scan(handle, &g_height, &g_slices, &a_height, &a_slices, &factor;
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		intout(0) = g_height
control(1) = 0		intout(1) = g_slices
control(2) = 0		intout(2) = a_height
control(3) = 0		intout(3) = a_slices
control(4) = 5		intout(4) = factor
control(5) = 24		
control(6) = handle		

V_ALPHA_TEXT (5-19H)

`v_alpha_text` outputs a specified string of alpha text to the printer. You must load a font before calling this function; see `vst_load_fonts` (77H) in Section 3. `v_alpha_text` is required only for printers.

The alpha text string is output at the current printer head position. All characters are output exactly as specified, with the following exceptions:

- The form-feed character (ADE value 12) has the same effect as a `v_form_advance` (5-14H) call.
- The two-character control sequences listed below invoke the described printer functions, if supported. DC2 refers to ADE value 18.

(DC2)0 -- Begin boldface
(DC2)1 -- End boldface
(DC2)2 -- Begin italics
(DC2)3 -- End italics
(DC2)4 -- Begin underscore
(DC2)5 -- End underscore

Input Arguments

<code>handle</code>	Device handle
<code>string</code>	ASCII character string

For the C language bindings, the string is byte-oriented and must be null terminated. For the Parameter Block (assembly) bindings, the string is word-oriented. `work_out[15]` of `vq_extnd` (66H) returns the maximum number of characters (size of `intin` array) allowed in the string. `vq_extnd` is described in Section 8.

Sample Call to C Language Binding

```
WORD  v_alpha_text();  
WORD  handle;  
BYTE  *string;
```

```
v_alpha_text(handle, string);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin = string	
control(1) = 0		
control(2) = 0		
control(3) = number of chars in string		
control(4) = 0		
control(5) = 25		
control(6) = handle		

VS_PALETTE (5-3CH)

This function allows the calling routine to select a color palette on the IBM® medium-resolution color screen. `vs_palette` returns the selected palette.

Input Arguments

<code>handle</code>	Device handle
<code>palette</code>	Color selection:
0	Use red, green, brown palette (default)
1	Use cyan, magenta, white palette

Output Arguments

<code>selected</code>	Selected palette
-----------------------	------------------

Sample all C Language Binding

```
WORD vs_palette();
WORD handle, palette;

selected = vs_palette(handle, palette);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = palette</code>	<code>intout(0) = selected</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 1</code>		
<code>control(5) = 60</code>		
<code>control(6) = handle</code>		

V_SOUND (5-3DH)

v_sound generates a tone of the specified frequency for the specified time interval.

Tone generation can be suppressed with vs_mute (5-3EH); see page 9-35.

Input Arguments

handle	Device handle
frequency	Tone frequency in hertz
duration	Tone duration in timer ticks

Sample Call to C Language Binding

```
WORD v_sound();
WORD handle, frequency, duration;

v_sound(handle, frequency, duration);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = frequency	
control(1) = 0	intin(1) = duration	
control(2) = 0		
control(3) = 2		
control(4) = 0		
control(5) = 61		
control(6) = handle		

VS_MUTE (5-3EH)

`vs_mute` sets, clears, or returns the current state of the muting flag. The muting flag controls tone generation.

Input Arguments

<code>handle</code>	Device handle
<code>action</code>	Action to perform:
	-1 Return state of muting flag
	0 Enable tone generation
	1 Disable tone generation

Output Arguments

<code>status</code>	Current state of muting flag (undefined for enable and disable calls):
	<code>zero</code> Tone generation enabled
	<code>nonzero</code> Tone generation disabled

Sample Call to C Language Binding

```
WORD vs_mute();
WORD handle,action;

status = vs_mute(handle,action);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = action</code>	<code>intout(0) = status</code>
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 0 for set</code>		
<code>1 for inquire</code>		
<code>control(5) = 62</code>		
<code>control(6) = handle</code>		

VT_RESOLUTION (5-51H)

This function sets the x and y resolution of the tablet. Resolution is specified independently for each axis in units of lines per inch. `vt_resolution` performs upper limit range checking on the `xres` and `yres` input arguments and returns the actual resolutions set.

Input Arguments

<code>handle</code>	Device handle
<code>xres</code>	Tablet x resolution in lines/inch
<code>yres</code>	Tablet y resolution in lines/inch

Output Arguments

<code>xset</code>	Actual x resolution set
<code>yset</code>	Actual y resolution set

Sample Call to C Language Binding

```
WORD vt_resolution();
WORD handle, xres, yres, xset, yset;

vt_resolution(handle, xres, yres, &xset, &yset);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = xres</code>	<code>intout(0) = xset</code>
<code>control(1) = 0</code>	<code>intin(1) = yset</code>	<code>intout(1) = yset</code>
<code>control(2) = 0</code>		
<code>control(3) = 2</code>		
<code>control(4) = 2</code>		
<code>control(5) = 81</code>		
<code>control(6) = handle</code>		

VT_AXIS (5-52H)

vt_axis sets the x and y axis resolution of the tablet. Resolution is specified independently for each axis in units of lines per axis. Since the resolution of the tablet must ultimately be expressed as an integral number of lines per inch, this vt_axis returns the actual axis resolution realized.

Input Arguments

handle	Device handle
xres	Tablet x resolution in lines
yres	Tablet y resolution in lines

Output Arguments

xset	Actual x resolution set
yset	Actual y resolution set

Sample Call to C Language Binding

```
WORD vt_axis();
WORD handle, xres, yres, xset, yset;

vt_axis( handle, xres, yres, &xset, &yset);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = xres	intout(0) = xset
control(1) = 0	intin(1) = yres	intout(1) = yset
control(2) = 0		
control(3) = 2		
control(4) = 2		
control(5) = 82		
control(6) = handle		

VT_ORIGIN (5-53H)

This function sets the x and y origin of the tablet.

Input Arguments

handle	Device handle
xorigin	x-coordinate of upper left corner of tablet origin in lines (tablet units)
yorigin	y-coordinate of upper left corner of tablet origin in lines (tablet units)

Sample Call to C Language Binding

```
WORD vt_origin();
WORD handle, xorigin, yorigin;

vt_origin(handle, xorigin, yorigin);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = xorigin	
control(1) = 0	intin(1) = yorigin	
control(2) = 0		
control(3) = 2		
control(4) = 0		
control(5) = 83		
control(6) = handle		

VQ_TDIMENSIONS (5-54H)

This function returns the x and y dimensions of the tablet in tenths of inches.

Input Arguments

handle Device handle

Output Arguments

xdimension Tablet x dimension in tenths of inches
ydimension Tablet y dimension in tenths of inches

Sample Call to C Language Binding

```
WORD  vq_tdimensions();
WORD  handle, xdimension, ydimension;

vq_tdimensions(handle, &xdimension, &ydimension);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5		intout(0) = xdimension
control(1) = 0		intout(1) = ydimension
control(2) = 0		
control(3) = 0		
control(4) = 2		
control(5) = 84		
control(6) = handle		

VT_ALIGNMENT (5-55H)

This function establishes an alignment between the physical coordinate axes of the tablet and the coordinate axes of, for example, a drawing affixed to the tablet surface.

vt_alignment is intended to account for inaccurate registration of a drawing with the physical coordinate axes, and might be used prior to tracing a drawing.

The input arguments are delta_x (dx) and delta_y (dy), which are the signed lengths of the sides of a triangle whose hypotenuse lies along the desired x or y axis of the drawing. If the hypotenuse of the triangle is defined by two points [x1, y1] and [x2, y2], delta_x is defined as (x2 - x1), while delta_y is defined as (y2 - y1).

Input Arguments

handle	Device handle
dx	Signed length of the x component of a triangle whose hypotenuse lies along the intended x or y axis
dy	Signed length of the y component of a triangle whose hypotenuse lies along the intended x or y axis

Sample Call to C Language Binding

```
WORD vt_alignment();
WORD handle, dx, dy;
```

```
vt_alignment(handle, dx, dy);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = dx	
control(1) = 0	intin(1) = dy	
control(2) = 0		
control(3) = 2		
control(4) = 0		
control(5) = 85		
control(6) = handle		

VSP_FILM (5-5BH)

`vsp_film` selects the film type and exposure time.

Input Arguments

handle	Device handle
index	Desired film type index number from 1 - n where n is the maximum available
	Use <code>vqp_filmname (5-5CH)</code> to obtain the name of the film associated with an index.
lightness	Lightness integer from -3 and 3. Each integer increase represents opening the aperture 1/3 of an f-stop. That is, -3 results in an exposure one half normal; 3 results in an exposure double normal.

Sample Call to C Language Binding

```
WORD vs_film();
WORD handle, index, lightness;

vsp_film(handle, index, lightness);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = index</code>	
<code>control(1) = 0</code>	<code>intin(1) = lightness</code>	
<code>control(2) = 0</code>		
<code>control(3) = 2</code>		
<code>control(4) = 0</code>		
<code>control(5) = 91</code>		
<code>control(6) = handle</code>		

VQP_FILMNAME (5-5CH)

This function returns the film name for the element specified by the calling routine. `vqp_filmname` returns the film name as a string up to 24 characters long, null terminated. If the specified element number (the index argument) is invalid, a null string is returned (control(4) is zero).

Input Arguments

<code>handle</code>	Device handle
<code>index</code>	Requested element number

Output Arguments

<code>status</code>	.TRUE. if <code>contrl[4] != 0</code> .FALSE. if <code>contrl[4] == 0</code>
<code>name</code>	Film name (ADE values), word-oriented for Parameter Block (assembly language) bindings, byte-oriented for C language binding

Sample Call to C Language Binding

```
BOOLEAN vqp_filmname();
BOOLEAN status;
WORD handle, index;
BYTE name[25]

status = vqp_filmname(handle, index, name);
```

Parameter Block Binding

Control	Input	Output
control(0) = 5	intin(0) = index	intout = name
control(1) = 0		
control(2) = 0		
control(3) = 1		
control(4) = number of chars in intout		
control(5) = 92		
control(6) = handle		

VSC_EXPOSE (5-5DH)

`vsc_expose` disables or enables the film exposure on some camera-type devices so that you can preview the frame prior to exposure. This function is limited to devices that support the preview capability.

Input Arguments

<code>handle</code>	Device handle
<code>state</code>	Exposure state set as follows:
<code>zero</code>	Disable exposure
<code>nonzero</code>	Enable exposure

Sample Call to C Language Binding

```
WORD vsc_expose();
WORD handle, state;

vsc_expose(handle, state);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = state</code>	
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = 1</code>		
<code>control(4) = 0</code>		
<code>control(5) = 93</code>		
<code>control(6) = handle</code>		

V_META_EXTENTS (5-62H)

`v_meta_extents` uses the values specified by the calling routine to update the extents information in the metafile header. The extents information can be used as a quick indication of the minimum rectangle that will bound all primitives output to the metafile. See Appendix B for a description of the metafile header and "Special Metafile Escapes."

If extent values are not defined when outputting to the metafile, all four bounding values default to zero.

Input Arguments

<code>handle</code>	Device handle
<code>min_x</code>	Minimum x value of the minimum bounding rectangle
<code>min_y</code>	Minimum y value of the minimum bounding rectangle
<code>max_x</code>	Maximum x value of the minimum bounding rectangle
<code>max_y</code>	Maximum y value of the minimum bounding rectangle

Sample Call to C Language Binding

```
WORD  v_meta_extents();
WORD  handle, min_x, min_y, max_x, max_y;
```

```
v_meta_extents(handle, min_x, min_y, max_x, max_y);
```

Parameter Block Binding

Control	Input
<code>control(0) = 5</code>	<code>ptsin(0) = min_x</code>
<code>control(1) = 2</code>	<code>ptsin(1) = min_y</code>
<code>control(2) = 0</code>	<code>ptsin(2) = max_x</code>
<code>control(3) = 0</code>	<code>ptsin(3) = max_y</code>
<code>control(4) = 0</code>	
<code>control(5) = 98</code>	
<code>control(6) = handle</code>	

V_WRITE_META (5-63H)

`v_write_meta` writes the `intin` and `ptsin` parameters to the metafile. The first word of the `intin` array should contain a sub-opcode that defines this information as a user-defined metafile item.

The VDI reserves sub-opcodes 0 through 100. The sub-opcode you use to define your metafile item must be numbered 101 or higher.

Input Arguments

<code>handle</code>	Device handle
<code>num_intin</code>	Length of <code>intin</code> array
<code>num_ptsin</code>	Number of input vertices
<code>intin</code>	User-defined information
<code>ptsin</code>	User-defined information

Sample Call of C Language Binding

```
WORD v_write_meta();
WORD handle, num_intin, num_ptsin, intin[num_intin],
      ptsin[num_ptsin];

v_write_meta(handle, num_intin, intin, num_ptsin, ptsin);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0) = sub-opcode</code>	
<code>control(1) = num_ptsin</code>	<code>intin = intin</code>	
<code>control(2) = 0</code>		
<code>control(3) = num_intin</code>	<code>ptsin = ptsin</code>	
<code>control(4) = 0</code>		
<code>control(5) = 99</code>		
<code>control(6) = handle</code>		

VM_FILENAME (5-64H)

When a metafile is first opened, its default name is GEMFILE.GEM and its default location is the current drive and directory. `vm_filename` renames a metafile from GEMFILE.GEM to the name specified by the calling routine. The metafile extension of GEM is maintained. The specified name can include a path and drive to locate the file somewhere other than on the current drive and directory.

Note: Call this function immediately after opening the metafile (see `v_opnwk` (1H) in Section 3) or it has no effect. `vm_filename` closes and deletes any open metafiles. Before this function is invoked, all metafile output is directed to GEMFILE.GEM.

Input Arguments

<code>handle</code>	Device handle
<code>filename</code>	Path/filename (null-terminated for C binding)

Sample Call to C Language Binding

```
WORD  vm_filename;
WORD  handle;
BYTE  *filename;
```

```
vm_filename(handle, filename);
```

Parameter Block Binding

Control	Input	Output
<code>control(0) = 5</code>	<code>intin(0 to n) = filename.</code>	
<code>control(1) = 0</code>		
<code>control(2) = 0</code>		
<code>control(3) = number of chars</code>		
	in filename	
<code>control(4) = 0</code>		
<code>control(5) = 100</code>		
<code>control(6) = handle</code>		

End of Section 9

GEM VDI Error Messages

Command line syntax error

Description: The VDI displays this message when the command line includes an illegal character, path, or drive specification.

Solution: Be sure that the command line conforms the operating system's command line conventions. Reenter the command line after correcting illegal entries.

Unable to find ASSIGN.SYS

Description: This message indicates that the VDI is unable to find the ASSIGN.SYS file in the specified location.

Solution: Place the ASSIGN.SYS file in the current directory.

Error reading ASSIGN.SYS

Description: The VDI cannot use the ASSIGN.SYS file because its format is incorrect.

Solution: Refer to Section 1 for the file's correct format.

Memory table corrupted

Description: This message indicates that the system's memory is corrupted.

Solution: Reboot your system.

Insufficient memory

Description: This message indicates that the VDI cannot satisfy your memory allocation request.

Solution: If your system has enough memory to run the VDI, reboot your system.

Invalid memory block address

Description: This message indicates that the system's memory is corrupted.

Solution: Reboot the system.

Drive specification not allowed in ASSIGN.SYS

Description: This message indicates that your ASSIGN.SYS file contains an illegal drive specification.

Solution: Edit the file to remove the drive specification. The correct format of an ASSIGN.SYS file is described in Section 1.

Illegal device id in ASSIGN.SYS

Description: The VDI displays this message when your ASSIGN.SYS file contains a device ID number that exceeds the range of values listed in Table 1-1.

Solution: Refer to Table 1-1 in Section 1 for the valid device ID numbers. Correct the ASSIGN.SYS file with your text editor.

Partial record found in ASSIGN.SYS

Description: This message indicates that your ASSIGN.SYS file contains a partial entry.

Solution: Check your ASSIGN.SYS file for incomplete device ID numbers or filenames. Refer to Section 1 for the correct ASSIGN.SYS file format.

Invalid filename encountered in ASSIGN.SYS

Description: Your ASSIGN.SYS file contains a filename which is either too long or contains illegal characters.

Solution: Locate the invalid filename within ASSIGN.SYS and correct it using the following guidelines:

- The filename must have eight or fewer characters.
- The first character in the filename must be alphabetic.
- The file's extension must be SYS.

ASSIGN.SYS file is empty

Description: The VDI displays this message to indicate that it has found an empty ASSIGN.SYS file.

Solution: Use your text editor to enter the necessary ASSIGN.SYS information. Section 1 describes the ASSIGN.SYS contents.

Corrupted driver file

Description: When the VDI has found, but is unable to use, a device driver specified in ASSIGN.SYS, it displays this message.

Solution: Use your distribution disk to make another copy of the device driver. Try to use the new copy. Contact your dealer if you are unable to use the device driver.

End of Appendix A

GEM VDI Metafile Format

This appendix describes the information output by the metafile driver and the operations it performs for various VDI functions.

Standard Metafile Item Format

Most of the function requests passed to the metafile driver result in the driver writing a standard metafile item to the metafile buffer. In a standard format metafile item, the control, integer, and vertex parameters are written to the metafile in the format shown in Table B-1.

Table B-1. Control, Integer and Vertex Parameters

Word	Value	Description
0	control[0]	opcode
1	control[1]	vertex count
2	control[3]	integer parameter count
3	control[5]	sub-opcode (or zero)
4...	ptsin[0-n]	input vertices (if provided)
n+4...	intin[0-m]	input integer (if provided)

Note if the vertex or integer parameter count is zero, nothing is output for the ptsin or intin information.

Standard Metafile Function Requests

Requests for the functions listed below cause the metafile driver to output a standard metafile item.

<u>Opcode (Decimal)</u>	<u>Function</u>
3	Clear Workstation, v_clrwk (3H)
4	Update Workstation, v_updwk (4H)
5-2	Exit Alpha Mode, v_exitcur (5-2H)
5-3	Enter Alpha Mode, v_entercur (5-3H)
5-20	Form Advance, v_form_adv (5-14H)
5-21	Output Window, v_output_window (5-15H)
5-22	Clear Display List, v_clear_disp_list (5-16H)
5-23	Output Bit Image File, v_bit_image (5-17H)
5-25	Output Printer Alpha Text, v_alpha_text (5-19H)
6	Polyline, v_line (6H)
7	Polymarker, v_pmarker (7H)
8	Text, v_qtext (8H)
9	Fill Area, v_fillarea (9H)
11-1	Bar GDP, v_bar (B-1H)
11-2	Arc GDP, v_arc (B-2H)
11-3	Pie GDP, v_pieslice (B-3H)
11-4	Circle GDP, v_circle (B-4H)
11-5	Ellipse GDP, v_ellipse (B-5H)
11-6	Elliptical Arc GDP, v_ellarc (B-6H)
11-7	Elliptical Pie Slice GDP, v_ellpie (B-7H)
11-8	Rounded Rectangle GDP, v_rbox (B-8H)
11-9	Filled, Rounded Rectangle GDP, v_rfbox (B-9H)
11-10	Justified Graphics Text GDP, v_justified (B-AH)
12	Set Character Height, Absolute Mode, vst_height (
13	Set Character Baseline Vector, vst_rotation (DH)
14	Set Color Representation, vs_color (EH)
15	Set Polyline Line Type, vsl_type (FH)
16	Set Polyline Line Width, vsl_width (10H)
17	Set Polyline Color Index, vsl_color (11H)
18	Set Polymarker Type, vsm_type (12H)
19	Set Polymarker Height, vsm_height (13H)
20	Set Polymarker Color Index, vsm_color (14H)
21	Set Text Font, vst_font (15H)
22	Set Text Color Index, vst_color (16H)

23	Set Fill Interior Style, vsf_interior (17H)
24	Set Fill Style Index, vsf_style (18H)
25	Set Fill Color Index, vsf_color (19H)
32	Set Writing Mode, vswr_mode (20H)
39	Set Graphic Text Alignment, vst_alignment (27H)
104	Set Fill Perimeter Visibility, vsf_perimeter (68H)
106	Set Graphic Text Special Effects, vst_effects (6AH)
107	Set Character Height, Points Mode, vst_point (6BH)
108	Set Polyline End Styles, vsl_ends (6CH)
112	Set User-defined Fill Pattern, vsf_udpat (70H)
113	Set User-defined Line Style Pattern, vsl_udsty (71H)
114	Fill Rectangle, vr_recfl (72H)
129	Set Clipping Rectangle, vs_clip (81H)

Nonstandard Metafile Items

Open Workstation - v_opnwk (1H)

v_opnwk initializes the metafile buffer and writes the metafile header to it. The device values normally returned by v_opnwk are returned. The format of the metafile header is described below.

Metafile Header Format

<u>Word</u>	<u>Description</u>
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 v_meta_extents (5-62H) in Section 9), 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 C, "Reserved Metafile Sub-opcodes.")

<u>Word</u>	<u>Description</u>
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 C.)
14	Bit image opcode flag: 0 No bit image opcode in file 1 Bit image opcode included in metafile Bits 1 - 15 must be zero.

Close Workstation - v_clswk (2H)

v_clswk appends an end-of-metafile marker, 0FFFFH (one word) to the metafile buffer. It then flushes the buffer and closes the metafile.

Special Metafile Escapes

There are three escape functions that provide special functions for metafiles. These functions are briefly described here; Section 9 provides more detailed information.

Update Metafile Extents - v_meta_extents (5-62H)

`v_meta_extents` updates words 4 - 7 in the metafile header to indicate the extents passed in the `ptsin` array.

Write Metafile Item - v_write_meta (5-63H)

`v_write_meta` writes the `intin` and `ptsin` parameters to the metafile. The first word of the `intin` array should contain a sub-opcode that defines what type of user-defined metafile item is being written. This special sub-opcode can be used by an application to identify the metafile item when it is read in. The VDI reserves sub-opcodes 0 through 100.

Change GEM VDI Filename - vm_filename (5-64H)

If any information currently exists in the metafile or metafile buffer, `vm_filename` flushes the buffer and closes the file. `vm_filename` reinitializes the metafile buffer and performs rudimentary file name validation. 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.

Inquire Functions**Inquire Addressable Character Cells - vq_chcells (5-1H)**

vq_chcells returns -1 in both intout(0), rows, and intout(1), columns, to indicate that cursor addressing is not possible.

Inquire Color Representation - vq_color (1AH)

vq_color returns -1 for the color index to indicate that no value is available.

Inquire Current Polyline Attributes - vql_attributes (23H)

vql_attributes returns the set values.

Extended Inquire - vq_extnd (66H)

vq_extnd returns the appropriate inquiry values.

End of Appendix B

Reserved Metafile Sub-opcodes

This appendix describes the metafile sub-opcodes reserved for two Digital Research GEM applications, GEM Output™ and GEM Draw™.

Metafile Sub-opcodes for GEM Output

The VDI reserves two sub-opcodes for sub-functions used by the GEM Output application:

<u>Sub-opcode</u>	<u>Sub-function</u>
0	Physical Page Size -- Defines how large a picture is to be rendered on the output page.
1	Coordinate Window -- Defines a newpage 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 output area. 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 Arguments

control(0)	5
control(1)	0
control(3)	3
control(5)	99
control(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 Arguments

control(2)	0
control(4)	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 `v_opnwk` (1H) in Section 3. For example, if raster coordinate space was specified when `v_opnwk` was invoked, 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 Arguments

<code>control(0)</code>	5
<code>control(1)</code>	0
<code>control(3)</code>	5
<code>control(5)</code>	99
<code>control(6)</code>	Device handle
<code>intin(0)</code>	Sub-opcode, 1
<code>intin(1)</code>	x-coordinate of lower left corner of window
<code>intin(2)</code>	y-coordinate of lower left corner of window
<code>intin(3)</code>	x-coordinate of upper right corner of window
<code>intin(4)</code>	y-coordinate of upper right corner of window

Output Arguments

<code>control(2)</code>	0
<code>control(4)</code>	0

Metafile Sub-opcodes for 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:

<u>Sub-opcode</u>	<u>Sub-function</u>
10	Group
49	Set No Line Style
50	Set Attribute Shadow On
51	Set Attribute Shadow Off
80	Start Draw Area Type Primitive
81	End Draw Area Type Primitive

Group

This sub-function indicates how many previous primitives are grouped. The number of preceding primitives in the group is passed to the sub-function in `intin(1)`.

Input Arguments

<code>control(0)</code>	5
<code>control(1)</code>	0
<code>control(3)</code>	2
<code>control(5)</code>	99
<code>control(6)</code>	Device handle
<code>intin(0)</code>	Sub-opcode number, 10
<code>intin(1)</code>	Number of preceding GEM Draw primitives in the group

Output Arguments

<code>control(2)</code>	0
<code>control(4)</code>	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 calls to `vsl_type (FH)`; see Section 5.

Input Arguments

- control(0) 5
- control(1) 0
- control(3) 1
- control(5) 99
- control(6) Device handle
- intin(0) Sub-opcode number, 49

Output Arguments

- control(2) 0
- control(4) 0

Set Attribute Shadow On

This sub-function is used by GEM Draw to indicate that all subsequent primitives that occur before the next Set Attribute Shadow Off sub-function should be ignored. This is because such primitives are used to draw a drop shadow for the first primitive immediately following the Set Attribute Shadow Off call.

Internally, GEM Draw assigns a shadowed attribute to the first primitive that follows the Set Attribute Shadow Off call 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 Arguments

control(0)	5
control(1)	0
control(3)	1
control(5)	99
control(6)	Device handle
intin(0)	Sub-opcode number, 50

Output Arguments

control(2)	0
control(4)	0

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 immediately following this sub-opcode.

Input Arguments

control(0)	5
control(1)	0
control(3)	1
control(5)	99
control(6)	Device handle
intin(0)	Sub-opcode number, 51

Output Arguments

control(2)	0
control(4)	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) that 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 Arguments

control(0)	5
control(1)	0
control(3)	1
control(5)	99
control(6)	Device handle
intin(0)	Sub-opcode number, 80

Output Arguments

control(2)	0
control(4)	0

End Draw Area Type Primitive

This sub-function indicates the end of an area type primitive block to GEM Draw.

Input Arguments

control(0)	5
control(1)	0
control(3)	1
control(5)	99
control(6)	Device handle
intin(0)	Sub-opcode number, 81

Output Arguments

control(2)	0
control(4)	0

End of Appendix C

Standard Keyboard

The 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

Table D-1. (Cont'd)

High Byte	Low Byte	Character
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
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

Table D-1. (Cont'd)

High Byte	Low Byte	Character
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
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

Table D-1. (Cont'd)

High Byte	Low Byte	Character
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

Table D-1. (Cont'd)

High Byte	Low Byte	Character
1A	7B	{
2B	7C	
1B	7D	}
29	7E	~
0E	7F	Rubout (DEL)
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

Table D-1. (Cont'd)

High Byte	Low Byte	Character
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
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

Table D-1. (Cont'd)

High Byte	Low Byte	Character
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
47	00	Home
47	37	Shift Home
52	00	Insert
52	30	Shift Insert
53	00	Delete
53	2E	Shift Delete

Table D-1. (Cont'd)

High Byte	Low Byte	Character
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	2D	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

Processor-Specific Data

8086-Specific Data

Registers and Interrupts

The application program passes the address of the Parameter Block to GEM VDI in registers DS:DX. Pass the VDI ID number, 1139 (0473H), in the register CX. The interrupt is EF.

Note: GEM VDI runs on Concurrent™ operating systems that support DOS calls of versions 2.0 and above.

Exchange Mouse Movement Vector

For 8086-based systems, 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 perform a RETURN FAR (RETF) instruction with the x,y position of the mouse the driver is to store in BX, CX. See vex_motv (7EH) in Section 7, "Input Functions," for information related to this function.

Exchange Button Change Vector

For 8086-based systems, the application code is invoked via a CALL FAR (CALLF) instruction with register AX containing the mouse button keys. Keys are encoded by the same rules that apply to the vq_mouse (7CH) function. When complete, the application-dependent code should perform a RETURN FAR (RETF) instruction with the mouse button state the driver is to store in AX. See vex_butv (7DH) for related information. Both vq_mouse and vex_butv are described in Section 7.

Exchange Cursor Change Vector

For 8086-based machines, the application-dependent code is invoked with a CALL FAR (CALLF) instruction. Upon entry, register BX contains the x position and the CX register the y position at which the cursor is to be drawn. If the application-dependent code does not draw its own cursor, a CALL FAR should be performed to the address returned in control(9) and control(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. See the description of `vex_curv` (7FH) in Section 7 for related information.

Exchange Timer Interrupt Vector

For 8086-based systems, the application-dependent code is invoked with a CALL FAR (CALLF) instruction. When complete, the application should perform a RETURN FAR (RETF) instruction.

See the description of `vex_timv` (76H) in Section 7 for related information.

68000-Specific Data

Registers and TRAPS

Pass the address of the Parameter Block to GEM VDI in one 32-bit register, D0.l. Register D1.w contains the GEM VDI ID number 115 (73H).

For CP/M-68K,TM 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 systems, 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 perform a RETURN FROM SUBROUTINE (RTS) instruction with the x,y position of the mouse the driver is to store in D0.w, D1.w.

See `vex_motv` (7EH) in Section 7, "Input Functions," for related information.

Exchange Button Change Vector

For 68000 processors, the application code is invoked via a JUMP TO SUBROUTINE (JSR) instruction and D0.w contains the mouse button keys. Keys are encoded by the same rules that apply to `vq_mouse` (7CH). When complete, the application-dependent code should perform a RETURN FROM SUBROUTINE (RTS) instruction with the mouse button state the driver is to store in D0.w.

See `vex_butv` (7DH) for related information. Both `vq_mouse` and `vex_butv` are described in Section 7.

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 control(9) and control(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.

Refer to the description of vex_curv (7FH) in Section 7 for related information.

Exchange Timer Interrupt Vector

The application-dependent code on 68000-based systems is invoked with a JUMP TO SUBROUTINE (JSR) instruction. When complete, the application should perform a RETURN FROM SUBROUTINE (RTS) instruction.

Refer to the description of vex_timv (76H) in Section 7 for related information.

End of Appendix E

Character Sets and Font Files

Character Sets

The system fonts provided with the VDI are illustrated in figures 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; see "Font Format," in this appendix.

DECIMAL VALUE	HEXA DECIMAL VALUE	0	16	32	48	64	80	96	112
0	0		▶	BLANK (SPACE)	0	@	P	'	p
1	1	↑	◀	!	1	A	Q	a	q
2	2	↓	◀▶	"	2	B	R	b	r
3	3	→	■	#	3	C	S	c	s
4	4	←		\$	4	D	T	d	t
5	5	■		%	5	E	U	e	u
6	6	◻		&	6	F	V	f	v
7	7	◆		'	7	G	W	g	w
8	8	✓		(8	H	X	h	x
9	9	☎)	9	I	Y	i	y
10	A	🔔		*	:	J	Z	j	z
11	B	🎵		+	;	K	[k	{
12	C	▲		,	<	L	\	l	!
13	D	▼		—	=	M]	m	}
14	E	▶		.	>	N	^	n	~
15	F	◀		/	?	O	_	o	△

Figure F-1. GEM VDI USASCII Character Set

DECIMAL VALUE	◆	128	144	160	176	192	208	224	240
◆	HEXA DECIMAL VALUE	8	9	A	B	C	D	E	F
0	0	Ç	É	á	ã	ij		α	≡
1	1	ü	æ	í	õ	IJ		β	±
2	2	é	Æ	ó	Ø	Φ		Γ	≥
3	3	â	ô	ú	ø			π	≤
4	4	ä	ö	ñ	œ			Σ	∫
5	5	à	ò	Ñ	Œ			σ	∫
6	6	â	û	ä	À			μ	÷
7	7	ç	ù	ó	Ã			τ	≈
8	8	ê	ÿ	¿	Õ			ϕ	○
9	9	ë	Ö	Γ	¨			θ	●
10	A	è	Ü	∩	´			Ω	·
11	B	ï	©	½	+			δ	√
12	C	î	£	¼	¢			§	¤
13	D	ì	¥	¡	©			φ	²
14	E	Ä	Œ	«	®			€	■
15	F	Å	ƒ	»	™			∩	BLANK (SPACE)

Figure F-2. GEM VDI International Character Set Extension

Font Format

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.

The format of the file is such that the low byte of a word occurs in memory before the high byte.

Font Header

The font header contains information that describes global aspects of the font. Font global aspects include information that applies to every character of the font, for example, the name of the face, the font size, and the minimum and maximum characters in the font. The format of the font header is shown in Table F-1.

Table F-1. Font Header Format

Byte Number	Description
0 - 1	Font identifier; see <code>vst_font</code> (15H) in Section 5
2 - 3	Font size in points
4 - 35	Font name; see <code>vqt_name</code> (82H) in Section 8
36 - 37	Lowest ADE value in the font, first character
38 - 39	Highest ADE value in the font, last character
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 <code>vqt_font_info</code> (83H) in Section 8
56 - 57	Right offset see <code>vqt_font_info</code> (83H) in Section 8
58 - 59	Thickening; this is the number of pixels by which to widen thickened characters
60 - 61	Underline size; this is the pixel-width of the underline

*Distances are measured relative to the character baseline and are always a positive value (magnitude rather than offset).

Table F-1. (Cont'd)

Byte Number	Description
62 - 63	Lightening mask; this mask is used to drop pixels out when lightening (usually 5555H)
64 - 65	Skewing mask; this mask determines when additional character rotation is required 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 Reserved -- must be zero 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)

Character Offset

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; this is indicated by the horizontal offset table bit in the Flags word (66) of the font header.

End of Appendix F

Bit Image File Format

A GEM VDI bit image file contains information that can 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 can be encoded in a variety of formats.

A bit image file has an extension of IMG.

Bit Image File Header

The bit image file header consists eight 16-bit words in which the high order byte precedes the low order byte. The header words are defined as listed in Table G-1.

Table G-1. Bit Image File Header Format

Word	Contents
0	Image file version number
1	Header length in words
2	Number of planes (source device bits per pixel)
3	Pattern definition length (number of bytes)
4	Source device pixel width (microns)
5	Source device pixel height (microns)
6	Scan line width (pixels)
7	Number of scan line items

Word 1 indicates how long the header is. Always check this value; future releases of GEM might have a longer header.

Word 3 defines the number of bytes used to describe pattern_run (pattern_run is described below). The value can range from 1 to 8. Note that this value is 2 for most bit image files taken from screen devices.

Bit Image File Data Format

The bit image data is composed of a series of scan line items. Word 7 in the file header indicates how many scan line items are present.

Each scan line item has two components:

- a vertical replication count
- encoded data for each plane

The vertical replication count is a word value formatted as follows:

<u>Byte</u>	<u>Contents</u>
0	NUL
1	NUL
2	FF Hex
3	Count

The count indicates how many identical scan lines are defined in this scan_line item.

The encoded data for each color plane follows the vertical replication count. The data is presented in the following order:

first plane -- red
second plane -- green
third plane -- blue
fourth plane -- grey

Data is always provided for all defined bit planes.

Note: The number of pixels described for each bit plane of a scan line is not necessarily the scan line width specified in the file header. Because the data is encoded in byte-wide packets (groups of eight pixels), the number of pixels actually described is always a multiple of eight and is never more than seven pixels wider than the scan line width.

Plane data is encoded in one of three modes: solid run, pattern run, or bit string.

A **solid_run** item contains a single byte that describes a state and the number of bytes for which that state is true. The high-order bit defines the state where:

- 1 = pixels on
- 0 = pixels off

The low-order seven bits define the run length. For example, to set a stream of 24 pixels (3 bytes) on, the encoded data is 83H. Similarly, to set a stream of 256 pixels (32 bytes) off, the encoded data is 20H.

A **pattern_run** item describes a set of pattern bytes and the number of times the pattern bytes should be repeated. The number of bytes in a pattern is defined in the bit image file header; typically, it is two for a screen device image. A **pattern_run** item is defined as follows:

<u>Byte</u>	<u>Contents</u>
0	NUL
1	Length of run
2	First byte of pattern
.	.
.	.
n	Last byte of pattern (n is defined by header word 3)

For example, a stream of 48 pixels (6 bytes) alternating red and blue with a pattern width of two is encoded as follows:

- Red plane item (hex values): 00 03 AA AA
- Blue plane item (hex values): 00 03 55 55
- Green plane item: The green bit plane in this example is a solid run so the encoded data is 06H (all pixels are off).

If a stream of pixels for a given plane cannot be encoded efficiently as a solid run or pattern run, it must be encoded as a **bit_string**. A **bit_string** item is defined as follows:

<u>Byte</u>	<u>Contents</u>
0	80 Hex
1	Byte count
2	First byte of bit string
.	
.	
n	Last byte of bit string

End of Appendix G

Index

8086-specific data, E-1

68000-specific data, E-3

A

Absolute mode, 5-19

Alpha cursor control, 9-6, 9-7,
9-8, 9-9, 9-10, 9-13,
9-17

Alpha Cursor Down, *v_curdown*,
9-7

Alpha Cursor Left, *v_curleft*, 9-9

Alpha cursor position, 9-17

Alpha Cursor Right, *v_currright*,
9-8

Alpha Cursor Up, *v_curup*, 9-6

Alpha text attribute, 9-15, 9-16

Alpha text string output, 9-14

Angle specifications, 4-15

Arc, *v_arc*, 4-17

Arrays, 1-5

Ascent line, 5-32

ASCII values, D-1

Assembly language routines,
2-11

ASSIGN.SYS, 1-4, 3-18

ASSIGN.SYS error messages,
A-1

Assignment table, 1-4

Attribute functions, 5-2

Attribute shadow, C-7, C-8

Axis scaling, 9-26

B

Background color, 3-4, 3-13

Bar, *v_bar*, 4-16

Baseline, 5-31

Baseline vector, 5-23

Bit image encoding modes, G-3

Bit image file, 9-26

Bit image file

color planes, G-2

data, G-1, G-2

file header, G-1

version, G-1

Bit string encoding, G-3

Bottom line, 5-32

C

Cell Array, *v_cellarray*, 3-8, 4-12

Change GEM VDI Filename,
vm_filename, 9-47

Character baseline, 5-19

Character cell, 5-19

Character codes, D-1

Character height, 3-9

Character offset table, F-6

Character rotation support, 8-15

Character scan line, F-4

- Character sets, F-1
- Character size, 5-21
- Character thickening, F-5
- Character top line, 5-19
- Character width, 3-9
- Circle, `v_circle`, 4-21
- Clear device, 3-13
- Clear Display List,
 - `v_clear_disp_list`, 9-25
- Clear Workstation, `v_clrwk`, 3-13
- Clipping rectangle, 3-20, 8-17
- Close events, 3-12
- Close Virtual Workstation,
 - `v_clsvwk`, 3-17
- Close Workstation, `v_clswk`, 3-12
- Color index, pixel, 6-9
- Color table index values, 3-4
- Command line error messages, A-1
- Command, GEMVDI, 2-15
- Contour Fill, `v_contourfill`, 4-29, 8-15
- Control, 2-11
- Control array, 1-5, 2-11
- Control functions, 3-1
- Coordinate scaling, 1-2
- Coordinate systems, 1-6, 6-6
- Coordinate window, C-2, C-3
- Copy Raster, Opaque,
 - `vro_cpyfm`, 6-10
- Copy Raster, Transparent,
 - `vrt_cpyfm`, 6-13
- CP/M-68K, E-3
- Cursor mask, 7-20

D

- Default values
 - character baseline rotation, 3-3
 - character height, 3-3
 - clipping, 3-3
 - color index, 3-4
 - cursor, 3-3
 - fill parameter visibility, 3-3
 - input mode, 3-3
 - line width, 3-3
 - marker height, 3-3
 - polyline end style, 3-3
 - text alignment, 3-3
 - text style, 3-3
 - user-defined fill pattern, 3-3
 - user-defined line style, 3-3
 - writing mode, 3-3
- Descent line, 5-32
- Device driver names, 1-4
- Device Drivers, 1-2
- Device handle, 1-3, 3-2
- Device ID number, 1-4, 3-5
- Device type, 3-9
- Device-specific information, 8-14
- Direct Alpha Cursor Address,
 - `vs_curaddress`, 9-13
- Disable/enable Film Exposure,
 - `vsc_expose`, 9-44

E

- Echo mode, 7-16, 7-18
- Ellipse, `v_ellipse`, 4-22

- Elliptical Arc, `v_ellarc`, 4-23
- Elliptical Pie, `v_ellpie`, 4-24
- End Draw Area Type Primitive, C-10
- End-of-metafile marker, B-5
- Enter Alpha Mode, `v_enter_cur`, 9-5
- Entering alpha mode, 9-5
- Entering graphics mode, 9-4
- Erase to End of Alpha Screen, `v_eeos`, 9-11
- Erase to End of Alpha Text Line, `v_eeol`, 9-12
- Error messages, A-1
- Escape functions, 9-1
- Exchange Button Change Vector (68000), E-3
- Exchange Button Change Vector (8086), E-1
- Exchange Button Change Vector, `vex_butv`, 7-28
- Exchange Cursor Change Vector (68000), E-4
- Exchange Cursor Change Vector (8086), E-2
- Exchange Cursor Change Vector, `vex_curv`, 7-32
- Exchange Mouse Movement Vector (68000), E-3
- Exchange Mouse Movement Vector (8086), E-1
- Exchange Mouse Movement Vector, `vex_motv`, 7-30
- Exchange Timer Interrupt Vector (68000), E-4
- Exchange Timer Interrupt Vector (8086), E-2

- Exchange Timer Interrupt Vector, `vex_timv`, 7-22
- Execute graphics commands, 3-14
- Exit Alpha Mode, `v_exit_cur`, 9-4
- Exposure time, 9-41
- Extended Inquire, `vq_extnd`, 8-14
- External fonts, F-4

F

- Face values, 5-25
- Fill interior style, 5-33
- Fill perimeter attribute, 4-10
- Fill Rectangle, `vr_recfl`, 4-30
- Filled Area, `v_fillarea`, 3-8, 4-10
- Filled Rounded Rectangle, `v_rfbox`, 4-26
- Film name, 9-42
- Film type, 9-41
- Flood fill, 4-29
- Font
 - ADE values, F-5
 - ascent line distance, F-5
 - bottom line distance, F-5
 - byte-orientation, F-6
 - character offset table, F-6
 - data, F-4
 - descent line distance, F-5
 - form height, F-4
 - form width, F-4
 - format, F-4
 - global aspects, F-4
 - half line distance, F-5
 - header, F-4
 - horizontal offset table, F-6,

- F-7
- identifier, F-5
- lightening mask, F-6
- raster area, F-4
- size in points, F-5
- skewing mask, F-6
- top line distance, F-5
- underline size, F-5
- Font identifiers, 3-18
- Fonts, 3-18, 3-19
- Form Advance, v_form_adv, 9-22
- Function parameter format, 2-11

G

- GDOS, 1-2
- GDP attributes, 3-8
- GDP ID numbers, 4-14
- GDPs, 3-7
- GEM Draw, C-4
- GEM file extension, 9-47
- GEM Output, C-1
- GEMFILE.GEM, 9-47
- GEMVDI command, 2-15
- Generalized Drawing Primitives,
3-7, 4-14
- Generate Tone, v_sound, 9-34
- Get Pixel, v_get_pixel, 6-9
- Graphic cursor control, 9-20,
9-21
- Group sub-function, C-5

H

- Half line, 5-32

- Handle, 3-2
- Handle
 - root device, 3-15
- Hard Copy, v_hardcopy, 9-19
- Hide Cursor, v_hide_c, 7-26
- Home Alpha Cursor, v_curhome,
9-10
- Horizontal offset table, F-6, F-7

I

- ID number
 - device, 3-5
- IMG file extension, G-1
- Initiating printer output, 3-14
- Input Choice, Request Mode,
vrq_choice, 7-13
- Input Choice, Sample Mode,
vsm_choice, 7-14
- Input functions, 7-1
- Input Locator, Request Mode,
vrq_locator, 7-5
- Input Locator, Sample Mode,
vsm_locator, 7-7
- Input modes, 7-1
- Input parameters array, 1-5,
2-11
- Input point coordinates array,
1-5, 2-11
- Input String, Request Mode,
vrq_string, 7-16
- Input String, Sample Mode,
vsm_string, 7-18
- Input Valuator, Request Mode,
vrq_valuator, 7-9
- Input Valuator, Sample Mode,

- vsm_valuator, 7-11
- Inquire Functions, 8-1
- Inquire Addressable Alpha Character Cells, vq_chcells, 9-3
- Inquire Camera Film Name, vqp_filmname, 9-42
- Inquire Cell Array, vq_cellarray, 8-4
- Inquire Character Cell Width, vqt_width, 8-22
- Inquire Color Representation, vq_color, 8-2
- Inquire Current Alpha Cursor Address, vq_curaddress, 9-17
- Inquire Current Fill Area Attributes, vqf_attributes, 8-10
- Inquire Current Font Information, vqt_font_info, 8-26
- Inquire Current Graphic Text Attributes, vqt_attributes, 8-12
- Inquire Current Polyline Attributes, vql_attributes, 8-6
- Inquire Current Polymarker Attributes, vqm_attributes, 8-8
- Inquire Font Name and Index, vqt_name, 8-24
- Inquire Input Mode, vqin_mode, 8-18
- Inquire Justified Graphics Text, vqt_justified, 8-29

- Inquire Printer Scan Heights, vq_scan, 9-29
- Inquire Tablet Dimensions, vq_tdimensions, 9-39
- Inquire Tablet Status, vq_tabstatus, 9-18
- Inquire Text Extent, vqt_extent, 8-19

- Interrupts
 - MS-DOS, E-1
 - PC DOS, E-1
- Intin, 2-11
- Intin array, 2-11
- Intout, 2-11

J

- Justified Graphics Text, v_justified, 4-27

K

- Keyboard definition, D-1

L

- Length of integer array, 2-14
- Length of intin array, 8-16
- Line style pattern word, 5-8
- Line type values, 5-8
- Line width, 3-10
- Load Fonts, vst_load_fonts, 3-18
- Locator terminator, 7-6

M

- Marker height, 3-10
- Marker width, 3-10
- Maximum number of vertices, 8-16
- Memory error messages, A-1
- Memory Form Definition Block, 6-2
- Memory requirements, 2-16
- Metafile, 1-3, 3-14
- Metafile
 - Metafile, 3-13
 - extent values, B-4
 - format, A-3
 - header format, B-4
 - item, B-1
 - reserved sub-opcodes, C-1
 - special escapes, B-6
 - Metafile bounding rectangle, 9-45
 - Metafile extent values, 9-45
 - Metafile item, 9-46
 - Metafiles, 9-45, 9-46, 9-47
 - Metafile header, 9-45
- MFDB, 6-2
- MFDB form format flag, 6-4
- Mouse
 - button state, E-1, E-3
 - position, E-1, E-3
- Mouse button state, 7-27
- Mouse form, 7-20
- MS-DOS registers and interrupts, E-1
- Muting flag, 9-35

N

- NDC, 1-6
- Normalized coordinate space, 1-2
- Normalized Device Coordinates, 1-6
- Number of available colors, 3-8
- Number of character heights, 3-6
- Number of choice devices, 3-9
- Number of colors, 8-15
- Number of columns, 9-3
- Number of faces, 3-7
- Number of hatch styles, 3-7
- Number of line types, 3-6
- Number of line widths, 3-7
- Number of locator devices, 3-8
- Number of marker sizes, 3-7
- Number of marker types, 3-7
- Number of patterns, 3-7
- Number of rows, 9-3
- Number of string devices, 3-9
- Number ofvaluator devices, 3-9

O

- Offset
 - character, F-6
 - horizontal, F-7
- Opcode
 - 1H, v_opnwk, 3-2
 - 2H, v_clswk, 3-12
 - 3H, v_clrwk, 3-13
 - 4H, v_updwk, 3-14
 - 5-1H, vq_chcells, 9-3

5-2H, v_exit_cur, 9-4
 5-3H, v_enter_cur, 9-5
 5-4H, v_curup, 9-6
 5-5H, v_curdown, 9-7
 5-6H, v_currigh, 9-8
 5-7H, v_curleft, 9-9
 5-8H, v_curhome, 9-10
 5-9H, v_eeos, 9-11
 5-AH, v_eeol, 9-12
 5-BH, vs_curaddress, 9-13
 5-CH, v_curtext, 9-14
 5-DH, v_rvon, 9-15
 5-EH, v_rvoff, 9-16
 5-FH, vq_curaddress, 9-17
 5-10H, v_eeol, 9-18
 5-11H, v_hardcopy, 9-19
 5-12H, v_dspcur, 9-20
 5-13H, v_rmcur, 9-21
 5-14H, v_form_adv, 9-22
 5-15H, v_output_window, 9-23
 5-16H, v_clear_disp_list, 9-25
 5-17H, v_bit_image, 9-26
 5-18H, vq_scan, 9-29
 5-19H, v_alpha_text, 9-31
 5-3CH, vs_palette, 9-33
 5-3DH, v_sound, 9-34
 5-3EH, vs_mute, 9-35
 5-51H, vt_resolution, 9-36
 5-52H, vt_axis, 9-37
 5-53H, vt_origin, 9-38
 5-54H, vq_tdimensions, 9-39
 5-55H, vt_alignment, 9-40
 5-5BH, vsp_film, 9-41
 5-5CH, vqp_filmname, 9-42
 5-5DH, vsc_expose, 9-44
 5-62H, v_meta_extents, 9-45
 5-63H, v_write_meta, 9-46
 5-64H, vm_filename, 9-47
 6H, v_pline, 4-4
 7H, v_pmarker, 4-6
 8H, v_gtext, 4-8
 9H, v_fillarea, 4-10
 AH, v_cellarray, 3-8, 4-12
 B-1H, v_bar, 4-16
 B-2H, v_arc, 4-17
 B-3H, v_pieslice, 4-19
 B-4H, v_circle, 4-21
 B-5H, v_ellipse, 4-22
 B-6H, v_ellarc, 4-23
 B-7H, v_ellpie, 4-24
 B-8H, v_rbox, 4-25
 B-9H, v_rfbox, 4-26
 B-AH, v_justified, 4-27
 CH, vst_height, 5-19
 DH, vst_rotation, 3-8, 5-23
 EH, vs_color, 3-8, 5-6
 FH, vsi_type, 5-8
 10H, vsi_width, 5-11
 11H, vsi_color, 5-12
 12H, vsm_type, 5-15
 13H, vsm_height, 5-17
 14H, vsm_color, 5-18
 15H, vst_font, 5-25
 16H, vst_color, 5-27
 17H, vsf_interior, 5-33
 18H, vsf_style, 5-35
 19H, vsf_color, 5-38
 1AH, vq_color, 8-2
 1BH, vq_cellarray, 8-4
 1CH, vrq_locator, 7-5
 1DH, vrq_valuator, 7-9
 1DH, vsm_valuator, 7-11
 1EH, vrq_choice, 7-13
 1EH, vsm_choice, 7-14

1FH, *vrq_string*, 7-16
1FH, *vsm_string*, 7-18
20H, *vswr_mode*, 5-3
21H, *vsin_mode*, 7-3
23H, *vql_attributes*, 8-6
24H, *vqm_attributes*, 8-8
25H, *vqf_attributes*, 8-10
26H, *vqt_attributes*, 8-12
27H, *vst_alignment*, 5-31
64H, *v_opnvwk*, 3-15
65H, *v_clsvwk*, 3-17
66H, *vq_extnd* (66H), 8-14
67H, *v_contourfill*, 4-29
68H, *vsf_perimeter*, 5-39
69H, *v_get_pixel*, 6-9
6AH, *vst_effects*, 5-28
6BH, *vst_point*, 5-21
6CH, *vsl_ends*, 5-13
6DH, *vro_cpyfm*, 6-10
6EH, *vr_trnfm*, 6-12
6FH, *vsc_form*, 7-20
70H, *vsf_udpat*, 5-40
71H, *vsl_udsty*, 5-10
72H, *vr_recl*, 4-30
73H, *vqin_mode*, 8-18
74H, *vqt_extent*, 8-19
75H, *vqt_width*, 8-22
76H, *vex_timv*, 7-22
77H, *vst_load_fonts*, 3-18
78H, *vst_unload_fonts*, 3-19
79H, *vrt_cpyfm*, 6-13
7AH, *v_show_c*, 7-24
7BH, *v_hide_c*, 7-26
7CH, *vq_mouse*, 7-27
7DH, *vex_butv*, 7-28
7EH, *vex_motv*, 7-30
7FH, *vex_curv*, 7-32

80H, *vq_key_s*, 7-34
81H, *vs_clip*, 3-20
82H, *vqt_name*, 8-24
83H, *vqt_font_info*, 8-26
84H, *vqt_justified*, 8-29

Opcodes, 1-3

Opcodes

for assembly language calls,
2-11

Open Virtual Workstation,
v_opnvwk, 3-15

Open Workstation, *v_opnwk*,
3-2, 8-14

Output Bit Image File,
v_bit_image, 9-26

Output Cursor Addressable
Alpha Text, *v_curtext*,
9-14

Output function fill attributes,
4-3

Output function line attributes,
4-2

Output function text attributes,
4-3

Output functions, 4-1

Output parameters array, 1-5,
2-11

Output point coordinates array,
1-5, 2-11

Output Printer Alpha Text,
v_alpha_text, 9-31

Output Window,
v_output_window, 9-23

P

- Parameter Block (PB), 2-11**
- Parameter Block**
 - address (68000), E-3**
 - address (8086), E-1**
 - format, 2-11**
- Pattern run encoding, G-3**
- Pattern word, 5-8**
- PB, 2-11**
- PC DOS registers and interrupts, E-1**
- Physical Page Size, C-2**
- Pie, v_pieslice, 4-19**
- Pixel aspect ration, 9-26**
- Pixel color index, 6-9**
- Pixel height, 3-6**
- Pixel height of printer head**
 - pass, 9-29**
- Pixel image, G-1**
- Pixel width, 3-6**
- Place Graphics Cursor, v_dspscur, 9-20**
- Plane data encoding, G-3**
- Plotter buffer, 3-14**
- Point size, 5-21**
- Points mode, 5-21**
- Polyline color index values, 3-4**
- Polyline, v_pline, 4-4**
- Polymarker type values, 5-15**
- Polymarker, v_pmarker, 4-6**
- Primitive ID numbers, 4-14**
- Printer buffer, 3-14**
- Printer display list, 9-22, 9-25**
- Printer head passes per page, 9-29**
- Printer scan height, 9-29**

- Ptsin, 2-11**
- Ptsin array, 2-11**
- Ptsout, 2-11**

R

- Raster area, 6-2**
- Raster area formats, 6-4**
- Raster Coordinates, RC, 1-7**
- Raster logic operations, 6-7**
- Raster operation writing modes, 6-13**
- Raster operations, 6-1**
- RC, 1-7**
- Registers**
 - 68000, E-3**
 - MS-DOS, E-1**
 - PC DOS, E-1**
- Remove Last Cursor, v_rmcur, 9-21**
- Renaming a metafile, 9-47**
- Replace mode, 5-3**
- Request input mode, 7-1**
- Required functions, 2-2**
- Return string, 7-16, 7-18**
- Reverse transparent mode, 5-4**
- Reverse Video Off, v_rvoff, 9-16**
- Reverse Video On, v_rvon, 9-15**
- RGB, 5-6**
- Root device handle, 3-15, 3-19, 7-30, 7-32**
- Rounded Rectangle, v_rbox, 4-25**

S

- Sample input mode, 7-1
- Sample Keyboard State Information, `vq_key_s`, 7-34
- Sample Mouse Button State, `vq_mouse`, 7-27
- Sample program, 2-17
- Scaling rectangle, 9-26
- Scan height division factor, 9-29
- Scan line, F-4
- Screen type, 8-14
- Seed fill, 4-29
- Select Camera Film Type and Exposure, `vsp_film`, 9-41
- Select Palette, `vs_palette`, 9-33
- Set Attribute Shadow Off, C-8
- Set Attribute Shadow On, C-7
- Set Character Baseline Vector, `vst_rotation`, 5-23
- Set Character Cell Height, Points Mode, `vst_point`, 5-21
- Set Character Height, Absolute Mode, `vst_height`, 5-19
- Set Clipping Rectangle, `vs_clip`, 3-20, 8-16
- Set Color Representation, `vs_color`, 3-8, 5-6
- Set Fill Color Index, `vsf_color`, 5-38
- Set Fill Interior Style, `vsf_interior`, 5-33
- Set Fill Perimeter Visibility, `vsf_perimeter`, 5-39
- Set Fill Style Index, `vsf_style`, 5-35
- Set Graphic Text Alignment, `vst_alignment`, 5-31
- Set Graphic Text Special Effects, `vst_effects`, 5-28
- Set Input Mode, `vsin_mode`, 7-3
- Set Mouse Form, `vsc_form`, 7-20
- Set No Line Style, C-6
- Set Polyline Color Index, `vsl_color`, 5-12
- Set Polyline End Styles, `vsl_ends`, 5-13
- Set Polyline Line Type, `vsl_type`, 5-8
- Set Polyline Width, `vsl_width`, 5-11
- Set Polymarker Color Index, `vsm_color`, 5-18
- Set Polymarker Height, `vsm_height`, 5-17
- Set Polymarker Type, `vsm_type`, 5-15
- Set Tablet Alignment, `vt_alignment`, 9-40
- Set Tablet Axis Resolution, `vt_axis`, 9-37
- Set Tablet Origin, `vt_origin`, 9-38
- Set Tablet Resolution, `vt_resolution`, 9-36
- Set Text Color Index, `vst_color`, 5-27
- Set Text Font, `vst_font`, 5-25
- Set User-defined Fill Pattern, `vsf_udpat`, 5-40
- Set User-defined Line Style,

- vsl_udsty, 5-10
- Set Writing Mode, vswr_mode, 5-3
- Set/Clear Muting Flag, vs_mute, 9-35
- Setting
 - color index, 5-6
 - exposure time, 9-41
 - fill area perimeter, 5-39
 - fill color, 5-38
 - fill interior style, 5-33
 - fill style, 5-35
 - line color, 5-12
 - line end style, 5-13
 - line type, 5-8
 - line width, 5-11
 - marker color, 5-18
 - marker height, 5-17
 - marker type, 5-15
 - text alignment, 5-31
 - text color, 5-27
 - text effects, 5-28
 - text font, 5-25
 - writing mode, 5-3
- Show Cursor, v_show_c, 7-24
- Solid run encoding, G-3
- Sound, 9-34
- Sound suppression, 9-35
- Special metafile escapes, B-6
- Stack requirements, 2-16
- Standard keyboard, D-1
- Start Draw Area Type Primitive, C-9
- Sub-opcodes, C-1
- Sub-opcodes
 - Coordinate Window, C-3
 - End Draw Area Type Primitive,

- C-10
- Group, C-5
- Physical Page Size, C-2
- Set Attribute Shadow Off, C-8
- Set Attribute Shadow On, C-7
- Set Draw Area Type Primitive, C-9
- Set No Line Style, C-6
- System fonts, F-4

T

- Table axes alignment, 9-40
- Table resolution, 9-36
- Terminator character, 7-6
- Text alignment, 5-31
- Text effect word, 5-28
- Text effects, 5-28, 8-15
- Text Rotation, vst_rotation, 3-8
- Text, v_gtext, 4-8
- Top line, 5-32
- Transform Mode, vr_trnfm, 6-12
- Transformation mode 0, 1-6
- Transformation mode 2, 1-7
- Transparent mode, 5-4
- TRAP instruction, 68000, E-3

U

- Undefined character symbol, 4-8, 4-27
- Unload Fonts, vst_unload_fonts, 3-19
- Update Metafile Extents, v_meta_extents, 9-45

Update Workstation, v_updwk,
3-14
USASCII character set, F-1
User-defined metafile item,
9-46

V

v_alpha_text, 9-31
v_arc, 4-17
v_bar, 4-16
v_bit_image, 9-26
v_cellarray, 3-8, 4-12
v_circle, 4-21
v_clear_disp_list, 9-25
v_clrwk, 3-13, 9-22, 9-25
v_clsvwk, 3-17
v_clswk, 3-12, B-5
v_contourfill (67H), 8-15
v_contourfill, 4-29
v_curdown, 9-7
v_curhome, 9-10
v_curleft, 9-9
v_currigh, 9-8
v_curtext, 9-14
v_curup, 9-6
v_dspcur, 9-20
v_eeol, 9-12
v_eeos, 9-11
v_ellarc, 4-23
v_ellipse, 4-22
v_ellpie, 4-24
v_enter_cur, 9-5
v_exit_cur, 9-4
v_fillarea, 3-8, 4-10
v_form_adv, 3-13, 9-22

v_get_pixel, 6-9
v_gtext, 4-8
v_hardcopy, 9-19
v_hide_c, 7-24, 7-26
v_justified, 4-27
v_meta_extents, 9-45, B-4, B-6
v_opnvwk, 3-15, 7-13, 7-14,
7-26
v_opnwk, 3-2, 7-13, 7-14, 7-26,
8-14, B-4, C-3
v_output_window, 9-23
v_pieslice, 4-19
v_pline, 4-4
v_pmarker, 4-6
v_pmarker attributes, 4-2
v_rbox, 4-25
v_recfl, 4-30
v_rfbox, 4-26
v_rmcur, 9-21
v_rvoff, 9-16
v_rvon, 9-15
v_show_c, 7-20, 7-24, 7-26
v_updwk, 3-14, 9-23
v_write_meta, 9-46, B-6
Valuator keys, 7-9
Values
ASCII, D-1
ASCII Decimal Equivalent, F-5
default attributes, 3-3
device type, 3-9
face, 5-25
fill interior style, 5-33
fill style, 5-36
keyboard, D-1
line ends, 5-13
line points, 5-13
line type, 5-8

- metafile extent, B-4
- MFDB format flag, 6-4
- pixel-to-color-index, 6-4
- polymarker type, 5-15
- text effects, 5-28
- VDI entry point, 1-5
- VDI ID number
 - 68000, E-3
 - 8086, E-1
- VDI keyboard assignments, D-1
- vex_butv, 7-28, E-1, E-3
- vex_curv, 7-32, E-2, E-4
- vex_motv, 7-30, E-1, E-3
- vex_timv, 7-22, E-2, E-4
- virtual workstations, 3-15
- vm_filename, 9-47, B-6
- vq_cellarray, 8-4
- vq_chcells, 9-3, 9-13, B-7
- vq_color, 8-2
- vq_curaddress, 9-17
- vq_extnd, 3-2, 3-20, 4-4, 4-6,
 - 4-8, 4-10, 4-27, 4-29,
 - 6-10, 8-14, 9-31
- vq_key_s, 7-34
- vq_mouse, 7-27, E-1, E-3
- vq_scan, 9-29
- vq_tabstatus, 9-18
- vq_tdimensions, 9-39
- vqf_attributes, 8-10
- vqin_mode, 8-18
- vql_attributes, 8-6, B-7
- vqm_attributes, 8-8
- vqp_filmname, 9-42
- vqt_attributes, 7-16, 7-18, 8-12
- vqt_extent, 8-19
- vqt_font_info, 8-26, F-5
- vqt_justified, 4-27, 8-29
- vqt_name, 8-24, F-5
- vqt_width, 8-22
- vr_trnfm, 6-4, 6-10, 6-12
- vro_cpyfm, 6-4, 6-6, 6-10
- vrq_choice, 7-13
- vrq_locator, 7-5
- vrq_string, 7-16
- vrq_valuator, 7-9
- vrqstring, 7-18
- vrt_cpyfm, 6-4, 6-6, 6-13
- vs_clip, 3-20, 4-3, 8-16
- vs_color, 3-8, 5-6, 8-2
- vs_curaddress, 9-13
- vs_mute, 9-35
- vs_palette, 9-33, 9-34
- vsc_expose, 9-44
- vsc_form, 7-20, 9-20
- vsf_color, 5-38
- vsf_interior, 5-33
- vsf_perimeter, 5-39
- vsf_style, 5-35
- vsf_udpat, 5-40
- vsin_mode, 7-3
- vsl_color, 5-12
- vsl_ends, 5-13
- vsl_type, 5-8, C-6
- vsl_udsty, 5-10
- vsl_width, 5-11
- vsm_choice, 7-14
- vsm_color, 5-18
- vsm_height, 5-17
- vsm_locator, 7-7, 9-20
- vsm_string, 7-18
- vsm_type, 5-15
- vsm_valuator, 7-11
- vsp_film, 9-41
- vst_alignment, 5-31, 8-16

- vst_color, 5-27
- vst_effects, 5-28
- vst_font, 5-25
- vst_height, 5-19
- vst_load_fonts, 3-18, 9-31
- vst_point, 5-21
- vst_rotation, 3-8, 5-23
- vst_unload_fonts, 3-19
- vswr_mode, 4-3, 5-3
- vt_alignment, 9-40
- vt_axis, 9-37
- vt_origin, 9-38
- vt_resolution, 9-36

W

- Workstation default attribute values, 3-3
- Write Metafile Item, v_write_meta, 9-46
- Writing a justified text string, 4-27
- Writing a text string, 4-8
- Writing modes, 5-3

X

- XOR mode, 5-4

GEM™ VDI Function Reference Table

Opcode		
Dec	Hex	Function and Binding
1	1	v_opnwk(work_in, &handle, work_out)
2	2	v_clswk(handle)
3	3	v_clrwk(handle)
4	4	v_updwk(handle)
5-1	5-1	vq_chcells(handle, &rows, &columns)
5-2	5-2	v_exit_cur(handle)
5-3	5-3	v_enter_cur(handle)
5-4	5-4	v_curup(handle)
5-5	5-5	v_curdown(handle)
5-6	5-6	v_currigh(handle)
5-7	5-7	v_curleft(handle)
5-8	5-8	v_curhome(handle)
5-9	5-9	v_eeos(handle)
5-10	5-A	v_eeol(handle)
5-11	5-B	vs_curaddress(handle, row, column)
5-12	5-C	v_curtext(handle, string)
5-13	5-D	v_rvon(handle)
5-14	5-E	v_rvoff(handle)
5-15	5-F	vq_curaddress(handle, &row, &column)
5-16	5-10	vq_tabstatus(handle)
5-17	5-11	v_hardcopy(handle)
5-18	5-12	v_dspcur(handle, x, y)
5-19	5-13	v_rmcur(handle)
5-20	5-14	v_form_adv(handle)
5-21	5-15	v_output_window(handle, xy)
5-22	5-16	v_clear_disp_list(handle)
5-23	5-17	v_bit_image(handle, filename, aspect, x_scale, y_scale, h_align, v_align, xy)
5-24	5-18	vq_scan(handle, &g_height, &g_slices, &a_height, &a_slices, &factor)
5-25	5-19	v_alpha_text(handle, string)
5-60	5-3C	vs_palette(handle, palette)
5-61	5-3D	v_sound(handle, frequency, duration)
5-62	5-3E	vs_mute(handle, action)
5-81	5-51	vt_resolution(handle, xres, yres, &xset, &yset)
5-82	5-52	vt_axis(handle, xres, yres, &xset, &yset)

Opcode		
Dec	Hex	Function and Binding
5-83	5-53	vt_origin(handle, xorigin, yorigin)
5-84	5-54	vq_tdimensions(handle, &xdimension, &ydimension)
5-85	5-55	vt_alignment(handle, dx, dy)
5-91	5-5B	vsp_film(handle, index, lightness)
5-92	5-5C	vqp_filmname(handle, index, name)
5-93	5-5D	vsc_expose(handle, state)
5-98	5-62	v_meta_extents(handle, min_x, min_y, max_x, max_y)
5-99	5-63	v_write_meta(handle, num_ints, ints, num_pts, pts)
5-100	5-64	vm_filename(handle, filename)
6	6	v_pline(handle, count, xy)
7	7	v_pmarker(handle, count, xy)
8	8	v_gtext(handle, x, y, string)
9	9	v_fillarea(handle, count, xy)
10	A	v_cellarray(handle, xy, row_length, el_per_row, num_rows, wr_mode, colors)
11-1	B-1	v_bar(handle, xy)
11-2	B-2	v_arc(handle, xc, yc, rad, sang, eang)
11-3	B-3	v_pieslice(handle, xc, yc, rad, sang, eang)
11-4	B-4	v_circle(handle, xc, yc, rad)
11-5	B-5	v_ellipse(handle, xc, yc, xrad, yrad)
11-6	B-6	v_ellarc(handle, xc, yc, xrad, yrad, sang, eang)
11-7	B-7	v_ellpie(handle, xc, yc, xrad, yrad, sang, eang)
11-8	B-8	v_rbox(handle, xy)
11-9	B-9	v_rfbox(handle, xy)
11-10	B-A	v_justified(handle, x, y, string, length, word_space, char_space)
12	C	vst_height(handle, height, &char_width, &char_height, &cell_wid &cell_height)
13	D	vst_rotation(handle, angle)
14	E	vs_color(handle, index, rgb)
15	F	vsl_type(handle, style)
16	10	vsl_width(handle, width)
17	11	vsl_color(handle, index)
18	12	vsm_type(handle, symbol)
19	13	vsm_height(handle, height)
20	14	vsm_color(handle, index)
21	15	vst_font(handle, font)

Opcode		
Dec	Hex	Function and Binding
22	16	vst_color(handle, index)
23	17	vsf_interior(handle, style)
24	18	vsf_style(handle, index)
25	19	vsf_color(handle, index)
26	1A	vq_color(handle, index, set_flag, rgb)
27	1B	vq_cellarray(handle, xy, row_len, num_rows, &el_used, &rows_used, &status, colarray)
28	1C	vrq_locator(handle, initx, inity, &xout, &yout, &term)
28	1C	vsm_locator(handle, initx, inity, &xout, &yout, &term)
29	1D	vrq_valuator(handle, val_in, &val_out, &term)
29	1D	vsm_valuator(handle, val_in, &val_out, &term, &status)
30	1E	vrq_choice(handle, in_choice, &out_choice)
30	1E	vsm_choice(handle, &choice)
31	1F	vrq_string(handle, length, echo_mode, echo_xy, string)
31	1F	vsm_string(handle, length, echo_mode, echo_xy, string, &status)
32	20	vswr_mode(handle, mode)
33	21	vsin_mode(handle, dev_type, mode)
35	23	vql_attributes(handle, attributes)
36	24	vqm_attributes(handle, attributes)
37	25	vqf_attributes(handle, attributes)
38	26	vqt_attributes(handle, attributes)
39	27	vst_alignment(handle, hor_in, vert_in, &hor_out, &vert_out)
100	64	v_opnvwk(work_in, &handle, work_out)
101	65	v_clsvwk(handle)
102	66	vq_extnd(handle, owflag, work_out)
103	67	v_contourfill(handle, x, y, index)
104	68	vsf_perimeter(handle, per_vis)
105	69	v_get_pixel(handle, x, y, &pel, &index)
106	6A	vst_effects(handle, effect)
107	6B	vst_point(handle, point, &char_width, &char_height, &cell_width, &cell_height)
108	6C	vsl_ends(handle, beg_style, end_style)
109	6D	vro_cpyfm(handle, wr_mode, xy, srcMFDB, desMFDB)
110	6E	vr_trnfm(handle, srcMFDB, desMFDB)
111	6F	vsc_form(handle, cur_form)

Opcode		
Dec	Hex	Function and Binding
112	70	vsf_udpat(handle, fill_pat, planes)
113	71	vsl_udsty(handle, pattern)
114	72	vr_recfl(handle, xy)
115	73	vqin_mode(handle, dev_type, &input_mode)
116	74	vqt_extent(handle, string, extent)
117	75	vqt_width(handle, character, &cell_width, &left_delta, &right_delta)
118	76	vex_timv(handle, tim_addr, &old_addr, &scale)
119	77	vst_load_fonts(handle, select)
120	78	vst_unload_fonts(handle, select)
121	79	vrt_cpyfm(handle, wr_mode, xy, srcMFDB, desMFDB, &index)
122	7A	v_show_c(handle, reset)
123	7B	v_hide_c(handle)
124	7C	vq_mouse(handle, &status, &px, &py)
125	7D	vex_butv(handle, usercode, &savecode)
126	7E	vex_motv(handle, usercode, &savecode)
127	7F	vex_curv(handle, usercode, &savecode)
128	80	vq_key_s(handle, &status)
129	81	vs_clip(handle, clip_flag, xy)
130	82	vqt_name(handle, element_num, name)
131	83	vqt_font_info(handle, &minADE, &maxADE, distances, &maxwidth, effects)
132	84	vqt_justified(handle, x, y, string, length, word_space, char_space, offsets)

