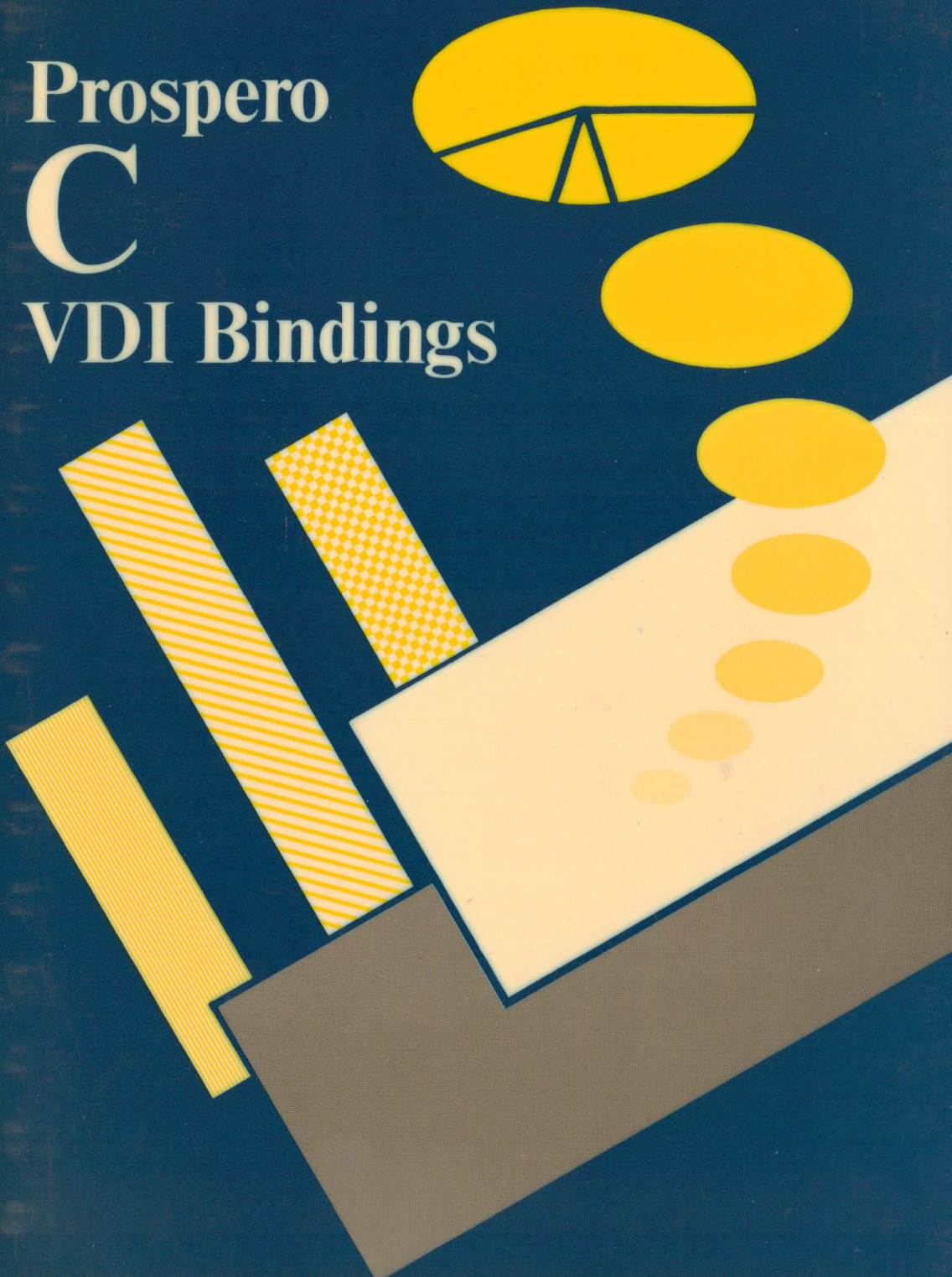
Prospero C VDI Bindings



Prospero
C
VDI Bindings

September 1990

Prospero Software
LANGUAGES FOR MICROCOMPUTER PROFESSIONALS

COPYRIGHT

DISCLAIMER

While every effort is made to ensure accuracy, Prospero Software cannot be held responsible for errors or omissions, and reserve the right to revise this document without notice.

TRADEMARKS

Acknowledgement is made for references in this manual to Microsoft and MS, which are trademarks of Microsoft Corp., to IBM, which is a trademark of International Business Machines Corp., to Apple and Macintosh, which are trademarks of Apple Computer Inc., to Digital Research and GEM, which are trademarks of Digital Research Inc., to Amstrad and Amstrad PC, which are trademarks of Amstrad Consumer Electronics plc, to Atari and Atari ST, which are trademarks of Atari Corp., to Motorola and MC68000, which are trademarks of Motorola Inc., and to Intel, which is a trademark of Intel Corp.

Prospero C, Pro Fortran-77, Prospero Fortran, Pro Pascal and Prospero Pascal are trademarks of Prospero Software.

Prospero Software, Inc.
100 Commercial Street, Suite 306
Portland, Maine 04101
U.S.A

Prospero Software Ltd.
190 Castelnau
London SW13 9DH
England

TABLE OF CONTENTS

1 INTRODUCTION TO GEM VDI

GEM VDI, Prospero C, and the Bindings

What is GEM VDI? Why should you want to use it? Why do you need bindings before you can do so, and why should these bindings require such a large amount of explanation in order to be used?

VDI stands for Virtual Device Interface – in other words, an interface to an imaginary, idealized device rather than to any particular real piece of hardware. Interfacing to an imaginary device rather than a real one makes more sense than might be immediately apparent, as all such imaginary devices can be made to look the same and work in the same way, whereas real devices always have different resolutions, numbers of colors, hardware arrangements, and codes required to drive them – come to that, some of them are screens, others are printers, others are graphics tablets, others are plotters and so on. If you write a program to drive a particular real device using its own codes, then it is more than likely that the program will need a lot of alterations before it will work on a different real device. So programs can be made more portable if they interface to a virtual device, then allow the virtual device to drive the real one.

GEM VDI is the successor to a previous device-independent graphics system known as GSX. It incorporates all the graphical operations that an application might be expected to require of any graphics device, and causes them to have the same effect (as far as possible) regardless of the actual device in question. Clearly some portion of GEM VDI will need to be device dependent, and different for every device that is attached – this portion is known as the device driver, and comes as a separate file so that drivers are only loaded into memory when output to a particular device is attempted. Drivers are available for most popular graphics hardware, and many are provided with the GEM Desktop when it is installed on a computer.

In order to call the routines provided by GEM VDI, a program must set up various memory areas with appropriate values, load a pointer to these memory areas into a register, then issue a particular software interrupt to cause the GEM VDI code to be entered. Such things are easily achieved when writing in assembler language, but to use GEM VDI from a high level language such as C, a number of functions callable from C have been provided which set up the relevant memory areas and pointers, then cause the interrupt to be issued. These bindings can then be linked with a C program to make an executable application.

The header file VDIBIND.H supplied with Prospero C contains definitions of various C types used in the GEM VDI bindings. Many of these are described in the functions where they are used – however, the type *WORD* is used widely throughout the VDI bindings :-

```
typedef short int WORD;        /* Used for all integer values */
```

for example

```
WORD handle;                   /* a workstation handle */
WORD coord;                    /* a screen coordinate */
WORD point[2];                 /* a coordinate pair */
WORD rect[4];
     /* two points, defining diagonally opposite corners of a rectangle */
```

The low level interface to GEM VDI need not usually concern the C programmer – all such details are handled by the bindings as transparently as possible. However the following may be of interest in some circumstances – further information can be found in the GEM Programmer's Toolkit, available from Digital Research.

GEM VDI is entered by a software interrupt, with the address of a parameter block contained in a register. This parameter block contains the addresses of 5 arrays through which information is passed to and from GEM VDI, and is declared as an external structure as follows :-

```
typedef WORD *LONG;

extern struct VDI_parmblock
    { LONG addr_control;
      LONG addr_intin;
      LONG addr_ptsin;
      LONG addr_intout;
      LONG addr_ptsout;
    } VDIparm;
```

For every call to GEM VDI, the fields of this structure must contain pointers to the arrays. Rather than set them up every time a GEM VDI call is made, they are initialized at program startup to point to five arrays, also declared as external, into which information is placed by each binding.

The five arrays are declared as follows :-

```
extern WORD VDI_control[12];
extern WORD VDI_intin[80];
extern WORD VDI_intout[80];
extern WORD VDI_ptsin[12];
extern WORD VDI_ptsout[12];
```

The above declarations are provided in VDIBIND.H in case you should want to use them in a program. Note that the above names must not be used for any other externally linked objects in a program which uses GEM VDI.

The `VDI_control` array is used for each GEM call to indicate which GEM VDI routine is required, and how many parameters are being passed in the other arrays in the following elements :-

0 – the function number required
1 – the number of points passed in the `VDI_ptsin` array
2 – the number of points returned in the `VDI_ptsout` array
3 – the number of values passed in the `VDI_intin` array
4 – the number of values returned in the `VDI_intout` array
5 – the sub-opcode for a GDP or Escape function (see sections 4 and 9)
6 – the device handle for the function

Some functions also use elements 7 to 11 of this array.

The `VDI_intin` array contains two-byte integer values to be used by the VDI. In general, all numbers, flags, characters and so on (other than coordinates) passed as parameters to the bindings are copied into the required position in this array. When strings are passed, the ASCII code of each character in turn is normally converted to a two-byte value and placed in this array. Thus the definitions above limit the maximum string which can be passed to 80 characters, of which one is the terminating null character. In certain circumstances it may be advantageous to increase the memory reserved for the `VDI_intin` array. This can be done by including in an application program definitions of the form

```
extern WORD VDI_control[12] = {0};
extern WORD VDI_intin  [nn] = {0};
extern WORD VDI_intout [80] = {0};
extern WORD VDI_ptsin  [12] = {0};
extern WORD VDI_ptsout [12] = {0};

extern struct VDI_parmblock VDIparm =
  { &VDI_control,
    &VDI_intin,  &VDI_ptsin,
    &VDI_intout, &VDI_ptsout };
```

Note that all five arrays and the VDIparm structure must be defined if you are going to define any, and none should be given smaller sizes than those given above unless you are very confident that the extra words are not required by any of the bindings you use.

The `VDI_intout` array is used to return two-byte values from VDI functions, which are then copied into the relevant parameters or function result by the bindings when values are to be returned.

The `VDI_ptsin` and `VDI_ptsout` arrays are used to pass coordinate pairs to and from GEM VDI – even elements contain x coordinates, with the corresponding y coordinate in the succeeding odd element. Where an array of points is passed (for example `v_pline` in section 4.1) the elements are not copied into this array, but the pointer to the array in the `VDIparm` record is temporarily adjusted. Thus the definition of these arrays as having 12 elements does not limit polylines to 6 vertices, but simply reflects the maximum number of points which the bindings need to copy into or out of these arrays.

2 USING GEM VDI

GEM VDI consists of a number of routines designed to be used for graphical output to devices, in such a way that programs can be written to be independent of the actual device being used. Thus the same function call can be expected to cause the same output whether the device in question is a screen, a plotter, or whatever. In order to achieve this, a separate driver is provided for each supported device, and loaded into memory when that device is to be used. One particular "device" which is supported is the Metafile – this is a record on disk of all the GEM VDI functions called, which therefore behaves as a generic description of a generated picture. The picture can then be sent to a device or used by another application at a later date.

GEM VDI considers each device to be a "workstation", and once the workstation is opened, and the relevant device driver loaded, all workstations will behave in a similar manner regardless of the device. It is only when opening a workstation that the program specifies which physical device is associated with that workstation – from then on the workstation is referred to only by its handle. Each workstation is independent of any other workstations currently in use, so that for example altering the line width or the clipping rectangle for one will not affect the subsequent output of another.

Two coordinate systems are available in GEM VDI – Normalized Device Coordinates (NDC) and Raster Coordinates (RC). The program must specify when a workstation is opened which coordinate system is to be used for that workstation. Normalized Device Coordinates are scaled so that the display area of any device has a coordinate range of 0 to 32767 in both x and y directions, with the origin at the bottom left hand corner (as in standard Cartesian coordinates). The advantage of using the NDC system is that it is independent of the resolution of the device – the point (16384, 16384) will be in the middle of a display whether it is a 640x400 screen or a 300 dots per inch laserwriter (resolution about 2500x3500 pixels). Raster coordinates correspond to the actual pixels of the device, and range from (0,0) in the top left hand corner to a device dependent maximum x and y coordinate, equal to the pixel resolution minus one. Raster coordinates are less portable than NDC coordinates, but more efficient to plot, as no scaling is required. If raster based operations, such as movement of blocks of pixels (see section 6) are to be used, the RC coordinate system will be the natural choice.

Rectangles in GEM VDI are always specified by giving the coordinates of two diagonally opposite corners, usually but not necessarily the top left and bottom right corners. This contrasts with the AES where rectangles are specified by giving the coordinates of the top left hand corner together with the rectangle's width and height. The coordinates are passed in an array[4] of *WORD*. For some routines, the coordinates of two such rectangles are specified in an array[8] of *WORD*.

3 VDI CONTROL FUNCTIONS

This section contains descriptions of the bindings for the various VDI control functions, in the following sub-sections.

Section	Function description	Binding name
3.1	Open Workstation	`v_opnwk`
	Open Virtual Workstation	`v_opnvwk`
3.2	Close Workstation	`v_clswk`
	Close Virtual Workstation	`v_clsvwk`
3.3	Clear Workstation	`v_clrwk`
3.4	Update Workstation	`v_updwk`
3.5	Load Fonts	`vst_load_fonts`
3.6	Unload Fonts	`vst_unload_fonts`
3.7	Set Clipping Rectangle	`vs_clip`

The functions in this section are mainly concerned with housekeeping. They are generally used at the beginning or end of a program or part of a program involving the use of a particular "workstation" – a GEM term for a display screen, printer, plotter or any other input or output device with which it has to work. Generally all workstations are treated in a consistent way; when an application opens a workstation GEM returns a number, called a handle, by which the workstation is identified. Whenever the application uses this workstation, it must give this handle as the first parameter. GEM thereafter interprets whatever you do in terms of the particular device's capabilities.

3.1 Open Workstation / Open Virtual Screen Workstation

v_opnwk
v_opnvwk

Before GEM can work with a device, such as a Screen, Plotter, Printer, Metafile, Camera, or Tablet it has to have access to information about the device. This information is placed in a special file on disk called a device driver. One of the secrets of GEM's success is that all the information that GEM needs to know about the particular printer, plotter or whatever is in one place; by changing the device driver alone GEM can then work with a different device – another manufacturer's printer, a laserprinter that can do both printing and plotting. In this way old software can run with new hardware by the simple addition of a new device driver. Device drivers are supplied by Digital Research or hardware manufacturers; a large number of drivers for popular peripherals are included with the GEM Programmer's Toolkit. These two functions are used to open the device driver files.

3.1.1 Definition

The Prospero C definitions of Open Workstation and Open Virtual Screen Workstation are:

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

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

3.1.2 Purpose

These functions load and initialize a device driver, and return various information about the driver and its capabilities, and an identifying "handle" to be used when referring to the workstation in future VDI calls. Open Virtual Screen Workstation allows several virtual workstations to be opened for a single screen device – each virtual screen workstation has access to the entire screen, with an independent set of attributes (line width, fill style, etc.) and clipping rectangle. Note that virtual screen workstations should not use sample mode for input functions (see section 7) as GEM cannot tell for which virtual workstation input is intended.

3.1.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
work_in	*WORD[11]*	**Initial attributes etc.**
		The parameter work_in is an array of 11 two-byte integers whose meanings are as follows :-
work_in[0]		Device number of required workstation, as listed in ASSIGN.SYS. Device numbers are allocated as follows :-
		1 – 10 Screen devices 11 – 20 Plotters 21 – 30 Printers 31 – 40 Metafile 41 – 50 Cameras 51 – 60 Tablets
work_in[1]		Initial line type (see section 5.3).
work_in[2]		Initial line color (see section 5.6).
work_in[3]		Initial marker type (see section 5.8).
work_in[4]		Initial marker color (see section 5.10).
work_in[5]		Initial text font (see section 5.13).
work_in[6]		Initial text color (see section 5.14).
work_in[7]		Initial fill interior style (see section 5.17).
work_in[8]		Initial fill style index (see section 5.18).
work_in[9]		Initial fill color index (see section 5.19).

`work_in[10]` Transformation flag :-

0 – use normalized device coordinates (NDC) to address the device, so that 0,0 refers to the lower left corner, and 32767,32767 refers to the upper right corner.

1 – reserved for future use.

2 – use raster coordinates (RC) to address the device, so that (0,0) refers to the top left corner, and (maxx,maxy) refers to the lower right corner, where maxx and maxy are the number of pixels supported in the horizontal and vertical directions, which can be obtained by examining the values returned in `work_out[0]` and `work_out[1]`.

For further information on suitable values, see the relevant set attributes function in section 5. The value 1 can safely be used for all of `work_in[1]` to `work_in[9]`.

`handle` *WORD ** **Device handle**

This points to the object used to return the device handle which GEM has allocated to the workstation just opened, which should be used as the first parameter to all subsequent GEM VDI calls for this workstation. A handle of zero indicates that the workstation could not be opened.

When opening a virtual screen workstation, the screen must have been opened as a physical workstation first, and the handle of the physical screen workstation should be passed to `v_opnvwk` in the object pointed to by this parameter, which will be modified by GEM before returning.

work_out	*WORD[57]*	**Workstation information** This parameter is an array of 57 two-byte integers into which the function will write information about the device which may be useful to the application :-
work_out[0]		Maximum addressable pixel in x axis (number of pixels – 1)
work_out[1]		Maximum addressable pixel in y axis
work_out[2]		0 – device capable of producing a precisely scaled image (e.g. a printer or plotter) 1 – not capable (e.g. a film recorder)
work_out[3]		Width of a pixel in microns (1000 microns = 1 mm)
work_out[4]		Height of a pixel in microns
work_out[5]		Number of character heights available (0 means continuous scaling)
work_out[6]		Number of line types available
work_out[7]		Number of line widths available (0 means continuous scaling)
work_out[8]		Number of marker types available
work_out[9]		Number of marker heights available (0 means continuous scaling)
work_out[10]		Number of faces (fonts) supported by device
work_out[11]		Number of fill patterns available
work_out[12]		Number of hatch styles available
work_out[13]		Number of predefined colors (the number of colors which the device can display simultaneously) 2 indicates a monochrome device.
work_out[14]		Number of GDPs (see section 4)

`work_out[15]` to `work_out[24]`

A list of those GDPs supported. There can be up to 10, as listed in section 4. The list is terminated by a value of -1, in which case any values of `work_out` between this value and `work_out[24]` are undefined.

`work_out[25]` to `work_out[34]`

A list describing which set of attributes is associated with each GDP in the above list :-

0 : line attributes
1 : marker attributes
2 : text attributes
3 : fill attributes
4 : no attributes

`work_out[35]` color capability (0 = no, 1 = yes)

`work_out[36]` text rotation capability (0 = no, 1 = yes)

`work_out[37]` area fill capability (0 = no, 1 = yes)

`work_out[38]` cell array capability (0 = no, 1 = yes)

`work_out[39]` number of colors available in palette :-

0 means continuous (>32767 colors)
2 means monochrome

`work_out[40]` Number of locator devices available (1 means keyboard only)

`work_out[41]` Number of valuator devices available (1 means keyboard only)

`work_out[42]` Number of choice devices available (1 means keyboard function keys only).

`work_out[43]` Number of string devices available

`work_out[44]` Workstation type :-

0 : output only
1 : input only
2 : input/output
3 : reserved
4 : metafile output

work_out[45]	Minimum character width in x axis
work_out[46]	Minimum character height in y axis
work_out[47]	Maximum character width in x axis
work_out[48]	Maximum character height in y axis.
	The above values are all in the current coordinate system. Note that these do not include inter-character or inter-line spacing.
work_out[49]	Minimum line width in x axis
work_out[50]	0 (not used)
work_out[51]	Maximum line width in x axis
work_out[52]	0 (not used)
work_out[53]	Minimum marker width in x axis
work_out[54]	Minimum marker height in y axis
work_out[55]	Maximum marker width in x axis
work_out[56]	Maximum marker height in y axis

3.1.4 Example

```
WORD work_in[11];
WORD work_out[57];
int  i;
WORD screen_handle, v_screen_handle;

main()
{
  for (i=0; i<10; i++)
    work_in[i] = 1;       /* Set initial attributes */
  work_in[10] = 2;        /* Use raster coordinates */
  v_opnwk(work_in, &screen_handle, work_out);
  if (screen_handle == 0 ) exit(3);        /* Abort */
  work_in[3] = 2;         /* different attributes */
  work_in[10] = 0;        /* Use NDC coordinates here */
  v_screen_handle = screen_handle;
  v_opnvwk(work_in, &v_screen_handle, work_out);
  /* The two workstations can now be used, with
    independent sets of attributes, clipping rectangles
    and so on */
  ...
}
```

3.2 Close Workstation / Close Virtual Workstation — `v_clswk` / `v_clsvwk`

When GEM has finished using a device opened with Open Workstation or Open Virtual Screen Workstation (see section 3.1) it must be closed, to make the workstation handle available for reuse by another application, and to release the memory in which the GEM VDI device driver stored the current settings of all the attributes. These two functions close a workstation previously opened with `v_opnwk` or `v_opnvwk`.

3.2.1 Definition

The Prospero C definitions of Close Workstation and Close Virtual Workstation are :

```
void v_clswk(WORD handle);
void v_clsvwk(WORD handle);
```

3.2.2 Purpose

This function is used to terminate output to a device, allowing the device to close properly and freeing the device handle for re-use by GEM. Closing a virtual screen workstation simply frees the workstation handle and its associated handle for reuse by GEM VDI. Closing a physical workstation will cause the device to tidy up as follows :-

Screen – return to alpha mode
Plotter – perform update
Printer – perform update
Metafile – write end of file and close the file
Camera – perform update

All virtual screen workstations associated with a physical screen workstation should be closed before using `v_clswk`.

3.2.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** This is the handle of the workstation or virtual workstation to be closed. No VDI calls using this handle should be made after a `v_clswk` or a `v_clsvwk` call.

3.2.4 Example

```
WORD screen_handle, other_handle;
WORD work_in[11];
WORD work_out[57];

main()
{
  v_opnwk(work_in, &screen_handle, work_out);
  other_handle = screen_handle;
  v_opnvwk(work_in, &other_handle, work_out);
  /* lots of GEM VDI calls using screen handle,
     other handle */
  /* close virtual workstation */
  v_clsvwk(other_handle);
  v_clswk(screen_handle); /* finished */
}
```

3.3 Clear Workstation `v_clrwk`

Clear Workstation is used to clear a workstation ready for further output. The precise effect depends upon the nature of the device.

3.3.1 Definition

The Prospero C definition of Clear Workstation is :

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

3.3.2 Purpose

This function is used to clear the specified workstation. The precise effect of this depends upon the nature of the device as follows :-

Screen – clear entire screen to background color – color index zero (see section 5).
Plotter – clear output buffer
Printer – clear output buffer and issue a form feed. See also `v_form_adv` in section 9.
Metafile – write Clear Workstation code to the metafile.
Camera – clear device to current background.

Note that Open Workstation (`v_opnwk`) will cause a device to be cleared when it is opened. However Open Virtual Screen Workstation (`v_opnvwk`) does not.

Note also that `v_clrwk` clears the printer or plotter buffer; normally a program should call `v_updwk` (see section 3.4) first to print or plot the contents of the buffer.

3.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device to be cleared.

3.3.4 Example

```
WORD screen_handle;

  /* Open screen device, do some output */
  v_clrwk(screen_handle);          /* Clear the screen */
```

3.4 Update Workstation v_updwk

When outputting graphics to a printer, the image can only be produced from the top of the page downwards. In order to print complex graphical images, the entire image must therefore be known before any output is made to the printer. GEM VDI printer drivers therefore maintain a buffer describing all output that has been sent to the printer, which is not actually transferred to paper until the program indicates that the image is complete by calling this function. This will output the image currently contained in the buffer, and advance to the next page. The contents of the buffer will not be erased, so that subsequent VDI output to the printer will add to the current buffer contents. For screen devices, all VDI output is immediately placed on the screen, and therefore this function has no effect.

3.4.1 Definition

The Prospero C definition of Update Workstation is:

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

3.4.2 Purpose

This function causes all pending output to be performed immediately. Plotter and printer devices have a buffer to which all output is sent, which will not be output to the device itself until this command is issued. Note that no form feed will be issued before the output is sent to the printer – `v_clrwk` (section 3.2) or `v_form_adv` (section 9.15) can be used for this purpose.

This function causes a metafile device to output an Update Workstation code.

3.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to be updated.

3.4.4 Example

```
WORD printer_handle;

  /* Open printer device, and output picture to it */
  ...
  /* Cause the picture to be printed */
  v_updwk(printer_handle);
```

3.5 Load Fonts `vst_load_fonts`

Load Fonts is used to load any external fonts which a device can use to output text. Most devices have a built in (system) font which remains available for use at all times, and additional external fonts, with a wider choice of letter styles, held in disk files so that they do not occupy memory unless they are required. Certain device drivers, notably printers, do not have a system font, and therefore require external fonts to be loaded before any GEM VDI text output may be sent to them.

The external fonts available for each device are listed in the ASSIGN.SYS file.

3.5.1 Definition

The Prospero C definition of Load Fonts is :

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

3.5.2 Purpose

This function is used to load the external fonts associated with the specified workstation's driver in the ASSIGN.SYS file. It is not necessary to load external fonts if the device's system font is all that is required – however, printer devices do not have system fonts, so a font must be loaded using `vst_load_fonts` before text can be output to a printer.

3.5.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose external fonts are to be loaded.
`select`	*WORD*	**Reserved** This parameter is reserved for future use. A value of zero should be passed.

3.5.4 Function Result

This function returns a two-byte integer value which indicates the number of extra fonts loaded.

3.5.5 Example

```
WORD printer_handle;
WORD number_of_fonts;
.
.
  /* Open printer device */
  .
  number_of_fonts = vst_load_fonts(printer_handle, 0);
  /* Select Swiss font */
  vst_font(printer_handle, 2);
  /* Text may now be output to the printer */
  .
  .
```

3.6 Unload Fonts `vst_unload_fonts`

External fonts loaded using the function `vst_load_fonts` (section 3.5) occupy a large amount of memory, especially as no means exists of only loading the ones that are required. When a font is no longer required Unload Fonts should be used to release memory allocated for the external fonts.

3.6.1 Definition

The Prospero C definition of Unload Fonts is:

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

3.6.2 Purpose

This function is used to free the memory allocated for external fonts loaded by the function `vst_load_fonts` (see section 3.5), when those fonts are no longer required. If fonts are shared between several virtual workstations, the memory will not be released until all virtual workstations have released it. Fonts are automatically released when a workstation is closed.

After this call has been made, the external fonts are no longer available. The device's system font(s) will still be available.

3.6.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose external fonts are to be unloaded.
`select`	*WORD*	**Reserved** This parameter is reserved for future use. A value of zero should be passed.

3.6.4 Example

```
WORD printer_handle;
WORD number_of_fonts;
.
.
  /* Open printer device */
  .
  number_of_fonts = vst_load_fonts(printer_handle, 0);
  /* Select Swiss font */
  vst_font(printer_handle, 2);
  /* Text may now be output to the printer */
  .
  /* release font memory */
  vst_unload_fonts(printer_handle, 0);
  /* Can still send graphics to printer, but no text */
```

3.7 Set Clipping Rectangle `vs_clip`

In GEM it is frequently useful to restrict graphics output to a particular portion of a workstation, for example to avoid overwriting the borders of a window or other windows. Set Clipping Rectangle provides this facility, and also enables or disables clipping of GEM VDI output.

3.7.1 Definition

The Prospero C definition of Set Clipping Rectangle is :

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

3.7.2 Purpose

This function is used to set the clipping rectangle of the specified workstation. All subsequent GEM VDI output will be clipped, so that only those portions which lie within the specified rectangle are output. It can also be used to enable or disable clipping.

The function `vq_extnd` (section 8.1) can be used to discover the current clipping rectangle (in GEM version 2.0 only)

3.7.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose clipping rectangle is being set.
clipflag	*WORD*	**Clipping enable/disable** If this is 1, clipping is enabled using the specified clipping rectangle. If 0, no clipping takes place (this is the default state when a workstation is opened).
pxyarray	*WORD[4]*	**Clipping rectangle** This parameter specifies the rectangle which is used to clip output to the device. The rectangle is defined by the coordinates in the current coordinate system (NDC or RC) of two diagonally opposite corners.

3.7.4 Example

```
WORD screen_handle;
WORD my_output_area[4];
.
  my_output_area[0] = 100;  /* x coord */
  my_output_area[1] = 100;  /* y coord */
                /* Top left hand corner of rectangle */

  my_output_area[2] = 400;  /* x coord */
  my_output_area[3] = 200;  /* y coord */
            /* Bottom right hand corner of rectangle */

  /* Switch clipping on */
  vs_clip(screen_handle, 1, my_output_area);
```

4 VDI OUTPUT FUNCTIONS

This section contains descriptions of the bindings for the GEM VDI output functions, in the following sub-sections.

Section	Function Description	Binding Name
4.1	Output Polyline	`v_pline`
4.2	Output Polymarker	`v_pmarker`
4.3	Output Text	`v_gtext`
4.4	Output Filled Area	`v_fillarea`
4.5	Output Cell Array	`v_cellarray`
4.6	Contour Fill	`v_contourfill`
4.7	Output Filled Rectangle	`vr_recfl`
4.8	Output Bar	`v_bar`
4.9	Output Arc Output Pieslice	`v_arc` `v_pieslice`
4.10	Output Circle	`v_circle`
4.11	Output Elliptical Arc Output Elliptical Pieslice	`v_ellarc` `v_ellpie`
4.12	Output Ellipse	`v_ellipse`
4.13	Output Rounded Rectangle Output Filled Rounded Rectangle	`v_rbox` `v_rfbox`
4.14	Output Justified Text	`v_justified`

VDI Output functions are provided to output graphical objects or text to a device. Several of these output functions are known as Generalized Device Primitives (GDPs). These need no special treatment by a programmer – the main difference is the manner in which the bindings have to be written. However, when a workstation is opened, information about the available GDPs is returned in the `work_out` array – see section 3.1 for further information. The GDPs are as follows:

1	`v_bar`	section 4.8
2	`v_arc`	section 4.9
3	`v_pieslice`	section 4.9
4	`v_circle`	section 4.10
5	`v_ellipse`	section 4.12
6	`v_ellarc`	section 4.11
7	`v_ellpie`	section 4.11
8	`v_rbox`	section 4.13
9	`v_rfbox`	section 4.13
10	`v_justified`	section 4.14

When using one of the GEM VDI output functions, the manner in which the output appears depends upon the current setting of the relevant attributes – these are described in detail in section 5. Each output function is affected by a particular set of attributes – there are four sets of attribute functions, which affect the output of lines, markers, filled areas and text. The set of attributes relevant for each function is specified in the section describing that function – the attribute set applying to each GDP (see above) is also specified in the values returned in `work_out` when a workstation is opened. All output functions are also affected by the current writing mode set by `vswr_mode` (section 5.1), and the current clipping rectangle set using `vs_clip` (section 3.7).

4.1 Output Polyline `v_pline`

GEM groups all graphics made up of lines into one function – the Polyline. This can be one line, two at any angle, three lines making up a triangle (or any line with three segments – the polyline need not form a closed figure). A large number of short lines may be used to draw what appears to be a curve, so that an entire graph may be drawn in one operation.

4.1.1 Definition

The Prospero C definition of Output Polyline is :

```
void v_pline(WORD handle, WORD count,
             WORD xyarray[]);
```

4.1.2 Purpose

This function is used to output a polyline to the specified device. A polyline consists of a series of straight lines joining the points contained in the array. The lines are drawn using the current line attributes and writing mode . For more information see the following functions :

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Line Type	`vsl_type`	(see section 5.3)
Set User Defined Line Style	`vsl_udsty`	(see section 5.4)
Set Line Width	`vsl_width`	(see section 5.5)
Set Line Color	`vsl_color`	(see section 5.6)
Set Line Ends	`vsl_ends`	(see section 5.7)

4.1.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device to which the polyline is to be output.

`count` *WORD* **Number of vertices**

This parameter specifies how many vertices make up the polyline. A single vertex (`count=1`) will not cause any output. The function `vq_extnd` (see section 8.1) can be used to discover the maximum number of vertices supported by this device.

`xyarray` *WORD[]* **Vertices of polyline**

This parameter contains the coordinates of each vertex of the polyline. Each vertex is specified by consecutive pairs of elements of `xyarray`: a line is drawn between each consecutive pair of vertices, up to the second last and last vertices. Thus `count-1` lines are drawn.

The array should contain twice as many elements as `count`.

All coordinates are in the current coordinate system (NDC or RC).

Lines of zero length are displayed as points.

4.1.4 Example

```
WORD my_line[14];
WORD screen_handle;
int  i;
  .
  .
  for (i = 0; i < 7; i++)
  {            /* A simple line graph */
    my_line[i*2] = 100 + 10*i;
    my_line[i*2 + 1] = 200 - i*i;
  }
     /* Output the line */
  v_pline(screen_handle, 7, my_line);
```

4.2 Output Polymarker `v_pmarker`

A marker is a dot, cross, triangle or other symbol that marks a point. A polymarker is a like a polyline (see section 4.1) except that it only shows markers at the points specified, rather than the lines joining the points. The most obvious use for Output Polymarker is to place markers on a line graph by using it after Output Polyline with the same `xyarray`. It can also be used to produce so-called scatter graphs, which only have points marked.

4.2.1 Definition

The Prospero C definition of Output Polymarker is :

```
void v_pmarker(WORD handle, WORD count,
               WORD xyarray[]);
```

4.2.2 Purpose

This function is used to output a number of markers to the specified device. The markers are drawn using the current marker attributes and writing mode as follows :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Marker Type	`vsm_type`	(see section 5.8)
Set Marker Height	`vsm_height`	(see section 5.9)
Set Marker Color	`vsm_color`	(see section 5.10)

4.2.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device to which the markers are to be output.
`count`	*WORD*	**Number of markers** This parameter specifies how many markers are to be output. The function `vq_extnd` (see section 8.1) can be used to discover the maximum number of markers supported by this device.

`xyarray` *WORD[]* **Vertices of polymarker**

This parameter contains the coordinates of each marker to be output. Each marker is specified by two consecutive elements, so `xyarray` has twice as many elements as `count`.

All coordinates are in the current coordinate system (NDC or RC).

4.2.4 Example

```
WORD marker_array[14];
WORD screen_handle;
int  i;
  .
  .
  for (i = 0; i < 7; i++)
  {   /* A simple scatter graph */
      marker_array[2*i] = 100 + 10*i;
      marker_array[2*i + 1] = 200 - i*i;
  }
   /* Output the markers */
  v_pmarker(screen_handle, 7, marker_array);
```

4.3 Output Text `v_gtext`

This routine is the basic text output function; using it a program can place any string of text at any position on a screen – or any other device. The size, color and style of the text can also be varied.

4.3.1 Definition

The Prospero C definition of Output Text is :

```
void v_gtext(WORD handle, WORD x, WORD y,
             const char *astring);
```

4.3.2 Purpose

This function is used to output a text string to the specified device. The text is output using the current text attributes – see the following functions :-

Set Text Height	`vst_height`	(see section 5.11)
Set Text Height (in points)	`vst_point`	(see section 5.11)
Set Character Baseline Vector	`vst_rotation`	(see section 5.12)
Select Character Font	`vst_font`	(see section 5.13)
Set Text Color	`vst_color`	(see section 5.14)
Set Text Effects	`vst_effects`	(see section 5.15)
Set Graphic Text Alignment	`vst_alignment`	(see section 5.16)

Note that if a device driver does not have a system font, an external font will have to be loaded and selected (see `vst_load_fonts` in section 3.5) before any text output can be made – this often applies to printer device drivers.

4.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the text is to be output.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates at which the text is to be output, in the current coordinate system. See the vst_alignment function in section 5.16 for information on how these coordinates relate to the position where the text is displayed.
astring	*const char* *	**Text** The text to be output.

4.3.4 Example

```
WORD screen_handle;
.
.
  /* Open screen workstation */
  v_gtext(screen_handle, 100, 50, "This is some text");
```

4.4 Output Filled Area v_fillarea

Filled polygons can be used to create graphics, area graphs, backgrounds or whatever, and is the fastest way to produce large areas of color or pattern. For rectangular or square areas, it will be easier to use `vr_recfl` or `v_bar` (see sections 4.7 and 4.8).

4.4.1 Definition

The Prospero C definition of Output Filled Area is:

```
void v_fillarea(WORD handle, WORD count,
                WORD xyarray[]);
```

4.4.2 Purpose

This function is used to output a filled polygon to the specified device. The polygon is defined by a set of vertices – there is no restriction on how complex the polygon may be (e.g. convex or concave, self-intersecting etc.). The polygon is filled using the current fill area attributes and writing mode – see the following functions for more information:-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

If the device is not capable of area fill, only the outline will be drawn in the current fill area color.

A polygon of zero area will be displayed as a dot if the fill perimeter visibility (see `vsf_perimeter` in section 5.20) is enabled, otherwise it will not be displayed at all.

4.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the polygon is to be output.
count	*WORD*	**Number of vertices** This parameter specifies how many vertices make up the polygon. A single vertex (count=1) will not cause any output. The function vq_extnd (section 8.1) can be used to discover the maximum number of vertices supported by this device.
xyarray	*WORD[]*	**Vertices of polygon** This parameter contains the coordinates of each vertex of the polygon. Each vertex is described by two consecutive elements in xyarray and so this array contains twice as many elements as count. All coordinates are in the current coordinate system (NDC or RC).

4.4.4 Example

```
WORD triangle[] = { 100, 200,
                    200, 200,
                    150, 300 };
WORD screen_handle;
  .
  .
   /* Output filled triangle */
  v_fillarea(screen_handle, 3, triangle);
```

4.5 Output Cell Array v_cellarray

This function can be thought of as a kind of knitting pattern; it works with a rectangular area, divided into equal size cells, and the color of each of the cells in the rectangle can be specified by an element of an array. It can thus be used to produced colored patterns as simple or as sophisticated as the program design requires.

4.5.1 Definition

The Prospero C definition of Output Cell Array is :

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

4.5.2 Purpose

This function is used to output a rectangular area to the device, divided into a number of cells each of whose color is specified by the corresponding element of an array. Each pixel in the rectangle will be set to the color index given by the array element corresponding to the cell which covers the pixel's centre.

Not all devices support cell array output – if a device does not support cell arrays, it will return a value of zero in `work_out[38]` when the device is opened with `v_opnwk` or `v_opnvwk` (section 3.1). If this function is used on a device which does not support cell arrays, the rectangle will be outlined using the current line attributes.

4.5.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the cell array is to be output.
xyarray	*WORD[4]*	**Rectangle** The rectangle to which the cell array is to be output, in the standard GEM VDI format of the coordinates of two diagonally opposite corners.
row_length	*WORD*	**Color array row length** The number of elements in each row of the color index array.
el_used	*WORD*	**Elements per row** The number of elements from each row which are to be used, and therefore the number of cells into which the rectangle is to be divided horizontally. This should be no greater than row_length – if it is less, some values at the end of each row will be ignored.
num_rows	*WORD*	**Color array row count** The rows to be used in the color index array. This should be no greater than the number of rows declared in the array declaration If it is less, any elements of the array after the last row used will be ignored. This parameter controls the number of rows of cells into which the rectangle will be divided.

`wrt_mode` *WORD* **Pixel writing mode**

The writing mode to be used when outputting the pixels. This function does not use the current writing mode as set by `vswr_mode` (see section 5.1), but requires the writing mode to be specified every time it is called. The possible values have the same meaning as in the function `vswr_mode` :-

1 – replace mode
2 – transparent mode
3 – XOR mode
4 – reverse transparent

See section 5.1 for further details.

`colarray` *WORD[]* **Color array**

The array containing the color index values to be used for each cell. The array should contain at least `row_length * num_rows` elements, each of type *WORD*. The individual cell colors can then be specified as `colarray[v * row_length + h]` where `v` and `h` give the row and column of the required cell.

4.5.4 Example

```
WORD colors[12];
WORD screen_handle;
int  i, j;
WORD rect[] = { 10, 10, 50, 70};
  .
  .
  for (i=0; i<3; i++)
    for (j=0; j<4; j++)
      colors[i*4 +j] = i + j;
                /* A rather dull pattern */

   /* Output 2 by 3 cells to rectangle */
  v_cellarray(screen_handle, rect, 4, 2, 3, 1,
              colors);
```

4.6 Contour Fill `v_contourfill`

A contour fill or 'flood fill' operation may be visualized as tipping a bucket of paint into a space in a drawing; the space, however large it is, is filled with an even color or pattern. There must be a solid boundary around the area to be flooded, or it will leak out and cover the whole drawing. Indeed a flood fill is a good way to add a background to all except closed areas. It is not generally useful for filling behind text because it does not fill inside closed letters such as a, o, p, d etc.

4.6.1 Definition

The Prospero C definition of Contour Fill is :

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

4.6.2 Purpose

This function is used to perform a 'flood fill' on the specified device. This is rather like pouring paint into a puddle – it will spread until it reaches a barrier. The barrier in this case is a pixel of the color specified by the parameter `index` – if these do not form a complete barrier then paint will leak out, and fill the entire device area.

The area is filled using the current fill area attributes and writing mode – see the following functions for more information:

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

This function is not supported by all devices – `vq_extend` (section 8.1) may be used to discover whether a device driver is capable of this.

4.6.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device where the flood fill is to be performed.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates of the point at which the flood fill is to start. All coordinates are in the current coordinate system (NDC or RC).
index	*WORD*	**Boundary color index** This is the color index which is to define the boundary at which the flood filling is to stop. A negative value will cause any pixel whose color does not match that of the starting point to be considered as part of the boundary.

4.6.4 Example

```
WORD screen_handle;
   .
   .
/* Output a few lines, arcs etc. to form a boundary*/
  v_contourfill(screen_handle, 100, 100, -1);
```

4.7 Output Filled Rectangle `vr_recfl`

As a simpler alternative to `v_fillarea` (section 4.4), this function can be used to output filled rectangular areas. The perimeter of the rectangle is never drawn.

4.7.1 Definition

The Prospero C definition of Output Filled Rectangle is :

```
void vr_recfl(WORD handle, WORD xyarray[4]);
```

4.7.2 Purpose

This function is used to output a filled rectangle to the specified device. The rectangle is filled using the current fill area attributes and writing mode, except for the fill perimeter visibility, which is always disabled. See the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

NB. This function is similar to `v_bar` (section 4.8), except that `v_bar` does use the fill perimeter visibility flag and can therefore be used to draw rectangles with the border marked.

4.7.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the rectangle is to be output.
xyarray	*WORD[4]*	**Rectangle** This parameter defines the rectangle to be output, in the standard VDI rectangle format of two diagonally opposite corners. All coordinates are in the current coordinate system (NDC or RC).

4.7.4 Example

```
WORD rect[4] = { 100, 200, 200, 400 };
WORD screen_handle;
.
.
  /* Set required fill attributes */
  ...
  /* Output filled rectangle */
  vr_recfl(screen_handle, rect);
```

4.8 Output Bar `v_bar`

A bar is a filled rectangle, used for such things as bar charts and so on. This function can be used to output such a rectangle using all the fill area attributes.

4.8.1 Definition

The Prospero C definition of Output Bar is :

```
void v_bar(WORD handle, WORD xyarray[4]);
```

4.8.2 Purpose

This function is used to output a filled rectangle to the specified device.

The function `v_bar` is a Generalized Drawing Primitive (GDP) with identifier 1. Its availability for a particular device may be determined by seeing whether its identifier 1 appears in the list of supported GDPs in `work_out[15]` to `work_out[24]` (cf. section 3.1).

The rectangle is filled using all the current fill area attributes and writing mode, including the fill perimeter visibility (unlike `vr_recfl`) – see the following functions for more information:-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

This function is similar to `vr_recfl` (section 4.7), except that `vr_recfl` does not use the fill perimeter visibility flag.

4.8.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the rectangle is to be output.
xyarray	*WORD[4]*	**Rectangle** This parameter defines the rectangle to be output, in the standard VDI rectangle format of two diagonally opposite corners. All coordinates are in the current coordinate system (NDC or RC).

4.8.4 Example

```
WORD rect[] = { 100, 200, 200, 400 };
WORD screen_handle;
.
.
  /* Set required fill attributes */
  ...
  /* Output filled rectangle */
  v_bar(screen_handle, rect);
```

4.9 Output Arc / Output Pieslice — v_arc / v_pieslice

Arcs and filled arc segments (called pieslices) are available for drawing pie charts and many other purposes.

4.9.1 Definition

The Prospero C definitions of Output Arc and Output Pieslice are:

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

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

4.9.2 Purpose

These functions are used to output respectively an arc or a pieslice to the specified device.

The functions `v_arc` and `v_pieslice` are Generalized Drawing Primitives (GDPs) with identifiers 2 and 3 respectively. Their availability for a particular device may be determined by seeing whether their identifiers 2 and 3 appear in the list of supported GDPs in `work_out[15]` to `work_out[24]` (cf. section 3.1).

The arc is drawn using the current line attributes and writing mode. For more information see the following functions :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Line Type	`vsl_type`	(see section 5.3)
Set User Defined Line Style	`vsl_udsty`	(see section 5.4)
Set Line Width	`vsl_width`	(see section 5.5)
Set Line Color	`vsl_color`	(see section 5.6)
Set Line End Style	`vsl_ends`	(see section 5.7)

The pieslice is drawn using the current fill attributes and writing mode. See the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

4.9.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the arc or pieslice is to be output.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates of the centre of the circle of which the arc or pieslice forms a part.
radius	*WORD*	**Radius** The radius of the circle of which the arc or pieslice forms a part, as measured along the x axis. All coordinates are in the current coordinate system (NDC or RC).
begang endang	*WORD* *WORD*	**Beginning angle** **End angle** These parameters define define the starting and ending angle of the arc or pieslice to be output. Angles are measured in tenths of a degree, starting at a line horizontally to the right and working anti-clockwise, so that vertically up is represented by 900. The arc is drawn working anti-clockwise from begang to endang. Values should be in the range 0 to 3600.

4.9.4 Example

```
WORD screen_handle;

  /* Set required fill and line attributes */
  /* Draw a circle, with the bottom half filled */
  v_arc(screen_handle, 200, 200, 50, 0, 1800);
  v_pieslice(screen_handle, 200, 200, 50, 1800, 3600);
```

4.10 Output Circle `v_circle`

This function is used to output a filled circle to a device. To output the outline of a circle using the current line attributes, a 360 degree arc can be drawn using the function `v_arc` (section 4.9).

4.10.1 Definition

The Prospero C definition of Output Circle is :

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

4.10.2 Purpose

This function is used to output a filled circle to the specified device.

The function `v_circle` is a Generalized Drawing Primitive (GDP) with identifier 4. Its availability for a particular device may be determined by seeing whether its identifier 4 appears in the list of supported GDPs in `work_out[15]` to `work_out[24]` (cf. section 3.1).

The circle is drawn using the current writing mode and fill area attributes. See the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

4.10.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the circle is to be output.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates of the centre of the circle.
radius	*WORD*	**Radius** The radius of the circle, as measured along the x axis. All coordinates are in the current coordinate system (NDC or RC).

4.10.4 Example

```
WORD screen_handle;
.
.
  /* Set required fill attributes */
  ...
  /* Draw a filled circle */
  v_circle(screen_handle, 200, 200, 50);
```

4.11 Output Elliptical Arc `v_ellarc`
Output Elliptical Pieslice `v_ellpie`

These functions can be used to draw an elliptical arc or filled arc segment. These might be used for drawing '3 dimensional' piecharts, and so on.

4.11.1 Definition

The Prospero C definitions of Output Elliptical Arc and Pieslice are :

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

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

4.11.2 Purpose

These functions are used to output respectively an elliptical arc or pieslice to the specified device.

The functions `v_ellarc` and `v_ellpie` are Generalized Drawing Primitives (GDPs) with identifiers 6 and 7 respectively. Their availability for a particular device may be determined by seeing whether their identifiers 6 and 7 appear in the list of supported GDPs in `work_out[15]` to `work_out[24]` (cf. section 3.1).

The arc is drawn using the current line attributes and writing mode. For more information see the following functions :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Line Type	`vsl_type`	(see section 5.3)
Set User Defined Line Style	`vsl_udsty`	(see section 5.4)
Set Line Width	`vsl_width`	(see section 5.5)
Set Line Color	`vsl_color`	(see section 5.6)
Set Line Ends	`vsl_ends`	(see section 5.7)

The pieslice is drawn using the current fill attributes and writing mode – see the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

4.11.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the elliptical arc or pieslice is to be output.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates of the centre of the ellipse of which the arc or pieslice forms a part.
xradius yradius	*WORD* *WORD*	**X radius** **Y radius** The x and y axis radii of the ellipse of which the arc or pieslice forms a part. All coordinates are in the current coordinate system (NDC or RC).
begang endang	*WORD* *WORD*	**Beginning angle** **End angle** These parameters define define the starting and ending angle of the elliptical arc or pieslice to be output. Angles re measured in tenths of a degree, starting at a line horizontally to the right and working anti-clockwise, so that vertically up is represented by 900. Values should be in the range 0 to 3600.

4.11.4 Example

```
WORD screen_handle;
.
.
  /* Set required fill and line attributes */
  ...
  /* Draw an ellipse, with the bottom half filled */
  v_ellarc(screen_handle, 200, 200, 100, 50, 0, 1800);
  v_ellpie(screen_handle, 200, 200, 100, 50,
           1800, 3600);
```

4.12 Output Ellipse `v_ellipse`

This function may be used to output a filled ellipse to a device. To output the outline of an ellipse using the current line attributes, the function `v_ellarc` (see section 4.11) may be used to draw a 360 degree elliptical arc. Only ellipses whose axes are parallel to the x and y axes are supported.

4.12.1 Definition

The Prospero C definition of Output Ellipse is :

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

4.12.2 Purpose

This function is used to output a filled ellipse to the specified device.

The function `v_ellipse` is a Generalized Drawing Primitive (GDP) with identifier 5. Its availability for a particular device may be determined by seeing whether its identifier 5 appears in the list of supported GDPs in `work_out[15]` to `work_out[24]`.

The ellipse is drawn using the current writing mode and fill area attributes. See the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

4.12.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the ellipse is to be output.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates of the centre of the ellipse.
xradius yradius	*WORD* *WORD*	**X radius** **Y radius** The x and y axis radii of the ellipse. All coordinates are in the current coordinate system (NDC or RC).

4.12.4 Example

```
WORD screen_handle;
.
.
  /* Set required fill attributes */
  ...
  /* Draw a filled ellipse */
  v_ellipse(screen_handle, 200, 200, 100, 50);
```

4.13 Output Rounded Rectangle `v_rbox`
Output Filled Rounded Rectangle `v_rfbox`

These functions are used to draw a rectangle, either filled or unfilled, with rounded corners.

4.13.1 Definition

The Prospero C definitions of Output Rounded Rectangle and Output Filled Rounded Rectangle are :

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

void v_rfbox(WORD handle, WORD xyarray[4]);
```

4.13.2 Purpose

These functions are used to output a rounded rectangle or a filled rounded rectangle to the specified device.

The functions `v_rbox` and `v_rfbox` are Generalized Drawing Primitives (GDPs) with identifiers 8 and 9 respectively. Their availability for a particular device may be determined by seeing whether their identifiers 8 and 9 appear in the list of supported GDPs in `work_out[15]` to `work_out[24]` (cf. section 3.1).

For `v_rbox`, the rectangle is drawn using the current line attributes and writing mode. See the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Line Type	`vsl_type`	(see section 5.3)
Set User Defined Line Style	`vsl_udsty`	(see section 5.4)
Set Line Width	`vsl_width`	(see section 5.5)
Set Line Color	`vsl_color`	(see section 5.6)
Set Line Ends	`vsl_ends`	(see section 5.7)

For `v_rfbox`, the rectangle is filled using the current fill area attributes and writing mode. See the following functions for more information :-

Set Writing Mode	`vswr_mode`	(see section 5.1)
Set Fill Interior Style	`vsf_interior`	(see section 5.17)
Set Fill Style	`vsf_style`	(see section 5.18)
Set Fill Color Index	`vsf_color`	(see section 5.19)
Set Fill Perimeter Visibility	`vsf_perimeter`	(see section 5.20)
Set User Defined Fill Pattern	`vsf_udpat`	(see section 5.21)

4.13.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the rectangle is to be output.
xyarray	*WORD[4]*	**Rectangle** This parameter defines the rectangle to be output, in the standard VDI rectangle format of two diagonally opposite corners. All coordinates are in the current coordinate system (NDC or RC).

4.13.4 Example

```
WORD rect[] = { 100, 200, 200, 400 };
WORD screen_handle;
.
.
   /* Set required fill or line attributes */
   ...
   /* Output rounded rectangle */
   v_rbox(screen_handle, rect);
```

4.14 Output Justified Text `v_justified`

This function may be used to output text to a device in such a way that the length of the line of text matches a requested value. Thus if several lines of text are output below each other, both left and right margins can be uniform.

4.14.1 Definition

The Prospero C definition of Output Justified Text is :

```
void v_justified(WORD handle, WORD x, WORD y,
                 const char *astring, WORD length,
                 WORD word_space, WORD char_space);
```

4.14.2 Purpose

This function is used to output a text string to the specified device, and attempt to justify it by expanding or contracting the inter-character and/or inter-word spacing so that the string is of a particular length. The x and y offsets of each character in a string output in this manner may be determined by using the `vqt_justified` function described in section 8.13.

The function `v_justified` is a Generalized Drawing Primitive (GDP) with identifier 10. Its availability for a particular device may be determined by seeing whether its identifier 10 appears in the list of supported GDPs in `work_out[15]` to `work_out[24]` (cf. section 3.1).

The text is output using the current writing mode and text attributes. For more information see the following functions :-

Set Text Height	`vst_height`	(see section 5.11)
Set Text Height (in points)	`vst_point`	(see section 5.11)
Set Character Baseline Vector	`vst_rotation`	(see section 5.12)
Select Character Font	`vst_font`	(see section 5.13)
Set Text Color	`vst_color`	(see section 5.14)
Set Text Effects	`vst_effects`	(see section 5.15)
Set Graphic Text Alignment	`vst_alignment`	(see section 5.16)

4.14.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the text is to be output.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates at which the text is to be output, in the current coordinate system. See the function `vst_alignment` in section 5.16 for information on how these coordinates relate to the position where the text is displayed.
astring	*const char **	**Text** The text to be output.
length	*WORD*	**Length of text string** The length, as measured along the x axis, to which the string is to be justified. (NB: NOT the length of the string in characters.)
word_space char_space	*WORD* *WORD*	**Modify word spacing** **Modify character spacing** These two flags determine whether GEM VDI is to modify the character spacing, the word spacing, or both when attempting to justify the string. A non-zero value is used if that option is wanted; else zero is used. If both are zero, then neither type of justification is permitted, and the text can not be justified.

4.14.4 Example

```
WORD screen_handle;

  v_justified(screen_handle, 100, 50,
              "This is a line of text", 200, 1, 1);
  v_justified(screen_handle, 100, 60,
              "This will line up with it", 200, 1, 1);
```

5 VDI ATTRIBUTE FUNCTIONS

This section contains descriptions of the VDI Attribute function bindings, in the following sub-sections.

Section	Function description	Binding name
5.1	Set Writing Mode	`vswr_mode`
5.2	Set Color Representation	`vs_color`
5.3	Set Line Type	`vsl_type`
5.4	Set User Defined Line Style	`vsl_udsty`
5.5	Set Line Width	`vsl_width`
5.6	Set Line Color	`vsl_color`
5.7	Set Line End Styles	`vsl_ends`
5.8	Set Marker Type	`vsm_type`
5.9	Set Marker Height	`vsm_height`
5.10	Set Marker Color	`vsm_color`
5.11	Set Text Height	`vst_height` `vst_point`
5.12	Set Character Baseline Vector	`vst_rotation`
5.13	Select Character Font	`vst_font`
5.14	Set Text Color	`vst_color`
5.15	Set Text Effects	`vst_effects`
5.16	Set Graphic Text Alignment	`vst_alignment`
5.17	Set Fill Interior Style	`vsf_interior`
5.18	Set Fill Style Index	`vsf_style`
5.19	Set Fill Color Index	`vsf_color`
5.20	Set Fill Perimeter Visibility	`vsf_perimeter`
5.21	Set User Defined Fill Pattern	`vsf_udpat`

VDI attribute functions are provided to control the style of all subsequent output to that workstation by the VDI Output functions described in section 4, and to alter the style attributes in force when the workstation is first opened. Most attribute functions return a value indicating what mode or style has been selected – this will be the same as that requested if available, otherwise what GEM considers to be the best substitute. Programs can test the result to check that all is well, discard it, or assign it to a variable to be referred to later.

5.1 Set Writing Mode `vswr_mode`

"Writing mode" determines how text, lines, patterns or whatever appears on the screen or other output device relative to what is already there. This is a concept which is perhaps most applicable to screens where reverse video is familiar. In paper and ink terms, the system may rub out what is already there before writing (mode 1); it may write on top of what is already there,and risk not being visible against a background of the same color (mode 2); it may write wherever there is nothing and invert anything already there so that the intersection of two black letters will be white (mode 3); and lastly may be reversed out of what is already there – typically white letters appearing in a dark or patterned background (mode 4).

5.1.1 Definition

The Prospero C definition of Set Writing Mode is :

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

5.1.2 Purpose

This function is used to select the writing mode which will determine how subsequent GEM VDI output interacts with what is already displayed. GEM VDI output primitives described in section 4 may be output in one of four writing modes, according to how the data being output (the mask – for example a fill pattern, a line style, or a bit pattern of some text) is to relate to the old color at each point, the current foreground color and the current background color. The available modes are as follows :-

`mode = 1` Replace mode

The new color is set to the foreground color wherever the mask is set, and to the background color wherever it is unset. This is the default mode when a workstation is opened. In this mode, text will appear on a strip of the background color. Each new output replaces what was previously displayed in the relevant pixels. This can be expressed as

new = (mask & foreground) | (~mask & background)

`mode` = 2 Transparent mode

Only pixels where the mask is set are affected, and set to the foreground color. In this mode, text would not appear on a strip of the background color, and what was previously displayed would still be visible through the gaps in the text. This can be expressed as

new = (mask & foreground) | (~mask & old)

`mode` = 3 XOR mode

This stands for exclusive OR – one or the other but not both. In this mode, all pixels where the mask is set are reversed. This has several uses – output in this mode will always have a visible effect regardless of the background, and if the same mask is output again the display will return to precisely the same as it was in the first place. This can be expressed as

new = mask ^ old

`mode` = 4 Reverse transparent mode

This is similar to transparent mode, except that only those pixels where the mask is unset are affected, being set to the foreground color. It can be expressed as

new = (mask & old) | (~mask & foreground)

This function affects the subsequent output of all GEM VDI output functions except `v_cellarray` (section 4.5).

5.1.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose writing mode is to be set.
`mode`	*WORD*	**Writing mode required** Selects which of the four output modes described above is to be used for subsequent output. If the value is not in the range 1 to 4, mode 1 (replace mode) will be selected.

5.1.4 Function Result

The value returned by this function is a two-byte integer indicating which writing mode has been selected – this will normally be the same as the value passed in the parameter mode, but may be different if the value is out of range or the device does not support that writing mode.

5.1.5 Example

```
WORD screen_handle;
int  i, j;
.
.
  /* Draw a creeping "Hello" */
  if (vswr_mode(screen_handle, 3) == 3)
                      /* XOR mode selected ok */
    for (i = 100; i <= 200; i++)
      {    /* Draw it */
      v_gtext(screen_handle, i, 100, "Hello");

      for (j = 0; j< 1000; j++);       /* Delay a bit */

      /* Restore display */
      v_gtext(screen_handle, i, 100, "Hello");
      }
  else
    v_gtext(screen_handle, 100, 100,
            "XOR mode not available");
```

5.2 Set Color Representation `vs_color`

This function is used to define the individual colors which are available for use on a screen or camera. On a device which uses a lookup table to determine the intensities of red, green and blue associated with each possible pixel value, this function can be used to set the exact shade of each available color index. On devices which have no lookup table, this function will have no effect. The number of colors which can be set in this way depends upon the device.

5.2.1 Definition

The Prospero C definition of Set Color Representation is :

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

5.2.2 Purpose

This function is used to select the intensity of red, green and blue with which a particular color index is to be displayed, and hence what precise shade it appears as. This will only have an effect on devices which support a color lookup table – this can be determined for a particular device using the function `vq_extnd` (section 8.1).

The function `vq_color` (section 8.2) can be used to discover the current color representation for a particular color index.

5.2.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose color representation is to be set.
index	*WORD*	**Color index** This specifies which color index is to be set. The background color is always referenced as color index 0. The maximum index is device dependent – it is returned as work_out[13] when the workstation is opened using v_opnwk or v_opnvwk (section 3.1).
rgb_in	*WORD[3]*	**Red green blue levels** This is an array of three two-byte integers, giving respectively the required intensity of red, green and blue for the given color index. The intensities are in tenths of percent (0 to 1000) – values less than 0 will be treated as 0, and values greater than 1000 will be treated as 1000.

5.2.4 Example

```
WORD screen_handle;
WORD my_rgb[3] = {1000, 0, 500};
.
.
  /* Make color index 3 purplish */

  vs_color(screen_handle, 3, my_rgb);
```

5.3 Set Line Type `vsl_type`

GEM VDI can produce lines of various types – solid, or dotted, or dashed, or more complex. This function determines which type of line is to be used for subsequent output.

5.3.1 Definition

The Prospero C definition of Set Line Type is :

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

5.3.2 Purpose

This function is used to select the line style which will be used subsequently when drawing VDI output primitives which use the line attributes. Note that some devices only support non-solid line patterns for lines of the default width – this can be checked using the function `vq_extnd` (section 8.1). When the workstation is opened, the initial line type is set to the value passed in `work_in[1]`.

The number of line styles available may depend on the device, but all should should support line styles 1 to 7 as follows :-

1 solid ----------
2 long dashed ---------- ----------
3 dotted --- --- --- ---
4 dash-dot ------- --- ------- ---
5 dashed -------- --------
6 dash-dot-dot ---- -- -- ---- -- --
7 application defined using `vsl_udsty` (section 5.4).

The availability of line styles with higher indices can be determined from the value returned in `work_out[6]` when the device is opened (see section 3.1).

This function affects the subsequent output of the functions `v_pline` (section 4.1), `v_arc` (section 4.9), `v_ellarc` (section 4.11) and `v_rbox` (section 4.13).

5.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose line style is to be set.
style	*WORD*	**Line style required** The required line style, as described above. If the specified index is out of range, a solid line will be selected.

5.3.4 Function Result

The value returned by this function is a two-byte integer indicating which line style has been selected – this will normally be the same as the value passed in the parameter style, but may be different if the value is out of range or the device does not support that line style. A solid line (style = 1) will be selected if the value passed is out of range.

5.3.5 Example

```
WORD screen_handle;
.
.
  /* Select dotted line */
  vsl_type(screen_handle, 3);
  /* Draw a circle with it */
  v_arc(screen_handle, 100, 100, 50, 0, 3600);
```

5.4 Set User Defined Line Style `vsl_udsty`

One of the available line styles that can be selected by the function `vsl_type` (section 5.3) is user defined. This function is used to set this extra style.

5.4.1 Definition

The Prospero C definition of Set User Defined Line Style is :

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

5.4.2 Purpose

This function is used to define the line style which is used when the user defined line style is selected by calling `vsl_type` with parameter `style` = 7. When the workstation is opened, the user defined line style is set to a solid line.

5.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose line style is to be set.
`pattern`	*WORD*	**Line pattern** This parameter is used to define which areas of the line are to be foreground and which background. It is treated as 16 one-bit values, each determining the state (1=foreground, 0=background) of that area of the line. The patterns corresponding to the predefined line styles are shown under `vsl_type` (section 5.3). The most significant bit of the pattern corresponds to the first pixel displayed.

5.4.4 Example

```
WORD screen_handle;
.
.
  /* Define a nearly solid line */
  vsl_udsty(screen_handle, 0xfffe);
  /* Select it */
  vsl_type(screen_handle, 7);
  /* Draw a circle with it */
  v_arc(screen_handle, 100, 100, 50, 0, 3600);
```

5.5 Set Line Width `vsl_width`

GEM VDI can produce lines of various widths – for screens the widths are 1, 3, 5 etc. pixels, depending on the screen. This function is used to set the width used for subsequent line output.

5.5.1 Definition

The Prospero C definition of Set Line Width is :

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

5.5.2 Purpose

This function is used to define the line width which is used when outputting GEM VDI primitives which use the line attributes. Note that when thickened lines are output, the solid line style may be used rather than that selected, and some writing modes may not be available – this may be determined using `vq_extnd` (section 8.1). When the workstation is opened, the line width is set to the default value.

This function affects the subsequent output of the functions `v_pline` (section 4.1), `v_arc` (section 4.9), `v_ellarc` (section 4.11) and `v_rbox` (section 4.13).

5.5.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose line width is to be set.
width	*WORD*	**Line width** The line width requested, as measured along the x axis in the current coordinate system. The line width which is closest to that requested, but not greater, will be selected. Line widths are odd numbers of pixel widths starting at one – the number available is returned in `work_out[7]` when the workstation is opened.

5.5.4 Function Result

The value returned indicates which line width has been selected – this may not necessarily be the same as that requested, but will never be greater.

5.5.5 Example

```
WORD screen_handle;
WORD graph[12];
.
.
  /* Select line width */
  vsl_width(screen_handle, 5);
  /* Draw a heavy graph */
  v_pline(screen_handle, 6, graph);
```

5.6 Set Line Color `vsl_color`

Lines can be drawn in any color the device will support; in the case of a monochrome device this will be black or white, while other devices may have more colors available.

5.6.1 Definition

The Prospero C definition of Set Line Color is :

```
WORD vsl_color(WORD handle, WORD color);
```

5.6.2 Purpose

This function is used to define the line color which is used when outputting GEM VDI primitives which use the line attributes. At least two colors (0 and 1) are always available. When the workstation is opened, the initial line color is set to the value passed in `work_in[2]`. The default color table indices are as follows :-

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

Values greater than 16 are device dependent. Color index 0 is defined to be the background color. If a device has a color lookup table, the color corresponding to each color index can be altered using `vs_color` (section 5.2).

This function affects the subsequent output of the functions `v_pline` (section 4.1), `v_arc` (section 4.9), `v_ellarc` (section 4.11) and `v_rbox` (section 4.13).

5.6.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose line color is to be set.
`color`	*WORD*	**Line color** The color index requested. Permissible values range from 0 to a device dependent maximum which can be discovered from the value of `work_out[39]` when the workstation is opened. If an out of range color index is requested, color index 1 will be selected. Color index 0 is the background color.

5.6.4 Function Result

The value returned is a two-byte integer indicating which line color index has been selected – this will normally be the same as that requested, unless it was out of range.

5.6.5 Example

```
WORD screen_handle;
WORD line_array[100];
.
.
  /* Set up line_array values first */

  /* Select line color */
  vsl_color(screen_handle, 1);
  /* Draw a graph */
  v_pline(screen_handle, 50, line_array);
```

5.7 Set Line End Styles `vsl_ends`

GEM VDI can draw lines with arrows at the ends, or with rounded or square cut ends. This function is used to set the end style used for the beginning and end of subsequent lines output.

5.7.1 Definition

The Prospero C definition of Set Line End Styles is :

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

5.7.2 Purpose

This function is used to define the style which is used to draw the ends of lines when drawing GEM VDI primitives which use the line attributes. When the workstation is opened, the initial line end style is squared.

This function affects the subsequent output of the functions `v_pline` (section 4.1), `v_arc` (section 4.9), `v_ellarc` (section 4.11) and `v_rbox` (section 4.13).

5.7.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose line end styles are to be set.
beg_style	*WORD*	**Line beginning style**
end_style	*WORD*	**Line end style** The required line end style for the beginning and end of each line respectively. The permitted values are as follows :- 0 Squared end 1 Arrow end 2 Rounded end The start and end of the line can be set to different styles if desired. If a value is requested which is out of range, squared ends will be selected. Squared and arrowed end styles finish at the end of the line in question – rounded ends use the end point of the line as the centre of the rounding.

5.7.4 Example

```
WORD screen_handle;
.
.
  /* Beginning arrowed, end rounded */
  vsl_ends(screen_handle, 1, 2);

  /* Draw a 90 degree arc, with a clockwise arrow */
  v_arc(screen_handle, 100, 100, 50, 0, 900);
```

5.8 Set Marker Type `vsm_type`

The function `v_pmarker` (section 4.2) can use a variety of different types of markers – 6 as standard and more on certain devices. This function can be used to select the marker style to be used, so that for example different lines on a graph use different marker styles.

5.8.1 Definition

The Prospero C definition of Set Marker Type is :

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

5.8.2 Purpose

This function is used to select the marker symbol which will be used for subsequent polymarker output. The initial polymarker symbol is specified by the value of `work_in[3]` when the workstation is opened.

All devices should support at least 6 polymarker symbols as follows :-

1 dot
2 plus
3 asterisk
4 square
5 diagonal cross
6 diamond

Marker symbols with index greater than 6 may be provided by some devices – these will be device dependent. The number of marker symbols supported by a device is returned in `work_out[8]` when the device is opened.

Note that marker symbol 1 is always output as the smallest dot which the device supports, and is not affected by the selected marker height (see `vsm_height` in section 5.9).

This function affects the subsequent output of the function `v_pmarker` (section 4.2).

5.8.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose marker symbol is to be set.
symbol	*WORD*	**Marker symbol required** The marker symbol to be used for subsequent polymarker output, as described above. If the requested value is out of range, marker symbol 3 (asterisk) will be selected.

5.8.4 Function Result

The value returned by this function is a two-byte integer indicating which marker symbol has been selected – this will normally be the same as the value passed in the parameter symbol, but may be different if the value is out of range or the device does not support that marker symbol. An asterisk (`symbol` = 3) will be selected if the value passed is out of range.

5.8.5 Example

```
WORD screen_handle;
WORD graph[12];
.
.
  /* Set up values of graph and required line style */

  /* Select square markers */
  vsm_type(screen_handle, 4);

  /* Draw a graph */
  v_pline(screen_handle, 6, graph);

  /* Mark the vertices */
  v_pmarker(screen_handle, 6, graph);
```

5.9 Set Marker Height `vsm_height`

Markers produced by `v_pmarker` (section 4.2) can have various sizes; this function is used to determine their height.

5.9.1 Definition

The Prospero C definition of Set Marker Height is :

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

5.9.2 Purpose

This function is used to define the marker height which is used when outputting polymarkers. When a workstation is opened, the polymarker height is set to the default value.

This function affects the subsequent output of the function `v_pmarker` (section 4.2).

5.9.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose marker height is to be set.
`height`	*WORD*	**Marker height** The marker height requested, as measured along the y axis in the current coordinate system. The available marker height which is closest to that requested, but not greater, will be selected. The number of available marker heights is returned in `work_out[9]` when the workstation is opened.

5.9.4 Function Result

The value returned is a two-byte integer indicating which marker height has been selected – this may not necessarily be the same as that requested, but will never be greater.

5.9.5 Example

```
WORD screen_handle;
WORD graph[12];
.
.
  /* Select marker height */
  vsm_height(screen_handle, 7);

  /* Draw a set of markers */
  v_pmarker(screen_handle, 6, graph);
```

5.10 Set Marker Color `vsm_color`

Polymarkers can be output in various colors. This function determines which is used for subsequent output.

5.10.1 Definition

The Prospero C definition of Set Marker Color is :

```
WORD vsm_color(WORD handle, WORD color);
```

5.10.2 Purpose

This function is used to define the marker color which is used when outputting polymarkers. At least two colors (0 and 1) are always available. When the workstation is opened, the initial marker color is set to the value passed in `work_in[4]`.

This function affects the subsequent output of the function `v_pmarker` (section 4.2).

5.10.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose marker color is to be set.
`color`	*WORD*	**Marker color** The color index requested. Permissible values range from 0 to a device dependent maximum which can be discovered from the value of `work_out[39]` when the workstation is opened. If an out of range color index is requested, color index 1 will be selected. Color index 0 is the background color.

5.10.4 Function Result

The value returned is a two-byte integer indicating which marker color index has been selected – this will normally be the same as that requested, unless it was out of range.

5.10.5 Example

```
WORD screen_handle;
WORD graph[100];
.
.
  /* Set up graph values */

  /* Select line color 3 and marker color 1 */
  vsl_color(screen_handle, 3);
  vsm_color(screen_handle, 1);

  /* Draw a graph */
  v_pline(screen_handle, 50, graph);
  /* Mark the points */
  v_pmarker(screen_handle, 50, graph);
```

5.11 Set Text Height

vst_height
vst_point

The size of characters to be output by GEM VDI may be varied in size using either of these routines. The size may be specified either in the current device coordinate system, or in point size, the type sizing system used by printers. Both routines return the sizes of a character and the cell it occupies. The function `vst_point` additionally returns the actual point size selected.

5.11.1 Definition

The Prospero C definitions of Set Text Height are :

```
void vst_height(WORD handle, WORD height,
                WORD *char_w, WORD *char_h,
                WORD *cell_w, WORD *cell_h);

WORD vst_point(WORD handle, WORD height,
               WORD *char_w, WORD *char_h,
               WORD *cell_w, WORD *cell_h);
```

5.11.2 Purpose

These routines are used to define the character height which will be used for subsequent graphic text output to the device. The two are very similar, the only difference being that `vst_height` requires the height to be given in device coordinates (NDC or raster coordinates), while `vst_point` requires the height to be specified in printer points, where one point is equivalent to 1/72 of an inch. Both return size information about the character set selected (in device coordinates); `vst_point` also returns as the function result the point size selected, so that an application can determine whether the requested point size was available.

These routines affect the subsequent output of the functions `v_gtext` (section 4.3) and `v_justified` (section 4.14).

5.11.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose text height is to be set.
height	*WORD*	**Requested character height** This parameter specifies the character height which the application wants. For vst_height, the value gives the height required from the character baseline to the top of the character cell, measured in y-axis units in the current device coordinate system (NDC or raster space). For vst_point, the value gives the height measured from the bottom of the character cell to the top of the character cell (i.e. the height of the character cell) in printer points (1/72 inch). In both cases, the available character set which is closest to but not greater than the requested size will be selected.
char_w char_h	*WORD* * *WORD* *	**Character width selected** **Character height selected** These parameters point to objects which return the width and height of the widest character in the selected character set, in the current device coordinate system. The height is measured from the character baseline to the top of the character cell, and the width does not include the left or right alignment deltas (see section 8.8).
cell_w cell_h	*WORD* * *WORD* *	**Character cell width selected** **Character cell height selected** These parameters point to objects which return the width and height of the character cell of the widest character in the selected character set, in the current device coordinate system.

5.11.4 Function Result

The value returned by `vst_point` gives the point size of the selected character set. This will always be less than or equal to the requested point size.

5.11.5 Example

```
WORD screen;
WORD w, h, cw, ch;
.
.
  vst_height(screen, 20, &w, &h, &cw, &ch);
  if (h != 20)
    { /* Character set 20 rasters high not available */
      ...
    }
  if (vst_point(screen, 12, &w, &h, &cw, &ch) != 12)
    { /* 12 point character set not available */
      ...
    }
```

5.12 Set Character Baseline Vector `vst_rotation`

Graphic text can be output in a normal horizontal way, or, on some output devices, at different angles. Drawing devices such as plotters are best at printing text at any angle; in pixel based devices such as screens, cameras, and matrix printers there is likely to be quality degradation at angles unless the pixel is small enough – in laserprinters it usually is. A device such as a daisy wheel printer will have no way of outputting a character at a different angle. This function serves to alter the angle of the line on which text is output.

5.12.1 Definition

The Prospero C definition of Set Character Baseline Vector is :

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

5.12.2 Purpose

This function is used to select the angle of the baseline parallel to which all subsequent graphic text will occur. Angles are specified in tenths of a degree, anti-clockwise from the horizontal, so that an angle of zero is normal default text output. Some devices may support only a limited number of angles, or none at all – this can be determined from the value in `work_out[36]` when a workstation is opened, or more precisely using the function `vq_extnd` (section 8.1), which returns a value of 0, 1 or 2 in `work_out[8]` according to whether the device supports no rotation, 90 degree increments only, or arbitrary angle rotations.

This function affects the subsequent output of the functions `v_gtext` (section 4.3) and `v_justified` (section 4.14).

5.12.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose text rotation vector is to be set.
angle	*WORD*	**Character baseline angle** This parameter specifies the character baseline vector angle requested by the application, in tenths of degrees anti-clockwise from horizontal. GEM VDI will select the best available angle of rotation, and return the rotation angle selected as the function result.

5.12.4 Function Result

The value returned gives the character rotation angle selected, in the same form as the `angle` parameter. This will be the best available approximation to the angle requested.

5.12.5 Example

```
WORD screen_handle;
.
.
  if (vst_rotation(screen_handle, 900) != 900)
    { /* Device can't write vertical text */
      ...
    }
```

5.13 Select Character Font `vst_font`

GEM has one built-in system font (typeface) for most devices – with the notable exception of printers. Any other font has to be loaded before use. This function is used to select a font for subsequent graphic text output.

5.13.1 Definition

The Prospero C definition of Select Character Font is :

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

5.13.2 Purpose

This function is used to select the font (also known in GEM VDI as a face) used for subsequent text output. Fonts are referred to by a font index number – font index 1 is the system font for a device (if it has one), and does not need to be loaded before use. Other fonts are external, and must be loaded using `vst_load_fonts` (section 3.5) before use. The indices for these external fonts may be determined using the `vqt_name` function (section 8.9). Indices in the range 1 to 17 are defined as follows :

1 – System font
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

This function affects the subsequent output of the functions `v_gtext` (section 4.3) and `v_justified` (section 4.14).

5.13.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose text font is to be selected.
font	*WORD*	**Requested text font** This parameter gives the index of the font requested. Note that some devices, particular printers, do not have system fonts. For these devices, font index 1 can not be selected, and an external font must be loaded and selected before graphic text can be output.

5.13.4 Function Result

The value returned gives the text font selected. This can be compared to the requested value to see if the font selection was successful, or simply ignored.

5.13.5 Example

```
WORD screen_handle;
.
.
  /* Select system font */
  vst_font(screen_handle, 1);
```

5.14 Set Text Color `vst_color`

Text may be output in various colors depending on the device in use. This function is used to select the color to be used for subsequent graphic text output.

5.14.1 Definition

The Prospero C definition of Set Text Color is :

```
WORD vst_color(WORD handle, WORD color);
```

5.14.2 Purpose

This function is used to select the color of subsequent graphic text output. Permissible values range from 0 to a device dependent maximum (always at least 1). If the value requested is beyond the maximum available index, color index 1 will be selected. When the workstation is opened, the initial text color is specified by the value in `work_in[6]`.

This function affects the subsequent output of the functions `v_gtext` (section 4.3) and `v_justified` (section 4.14).

5.14.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose text color is to be set.
`color`	*WORD*	**Requested text color** This parameter gives the index of the color requested.

5.14.4 Function Result

The value returned gives the color index selected. This will be the same as that requested if it was in range, otherwise color index 1 will be selected.

5.14.5 Example

```
WORD screen_handle;
WORD color;
int  i;
.
.
  i = 2;
  do {
    color = vst_color(screen_handle, i++);
    /* Output some text */
  } while (color != 1);    /* in every color available
                              except index 0 */
```

5.15 Set Text Effects `vst_effects`

GEM can alter a font before using it for output. These alterations are, in printer's terms, bold, light, italic, underline, outline or shadow. The success of these alterations from a graphic point of view is variable as they are mathematical transformations without any design input. Text Effects should not be confused with selecting a different version of a font such as Swiss Light Italic; in this case the letters of the font are individually designed. This function is used to set the special effects to be used for subsequent graphic text output.

5.15.1 Definition

The Prospero C definition of Set Text Effects is :

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

5.15.2 Purpose

This function is used to select the text effects to be used for subsequent graphic text output. The effects available are thickened, light, skewed, underlined, outlined, shadowed, or any combination. The required effects are specified using a bitmap – the required state (selected or not selected) of each effect is determined by the value (0 or 1) of the corresponding bit in the `effect` parameter. The easiest way to use such a bitmap is to declare a constant with just the relevant bit set for each effect, then to combine the constants of the required effects using the OR (`|`) operator. Suitable constants would be as follows :-

```
#define normal      0x0000
#define thickened   0x0001
#define light       0x0002
#define skewed      0x0004
#define underlined  0x0008
#define outlined    0x0010
#define shadowed    0x0020
```

The function returns a similar bitmap indicating which of the requested effects were available – this can be compared to the requested value to discover which effects were not available.

This function affects the subsequent output of the functions `v_gtext` (section 4.3) and `v_justified` (section 4.14).

5.15.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose text effects are to be set.
effect	*WORD*	**Requested text effects** This parameter specifies the requested effects, one bit per effect, where a bit value of 1 indicates the effect is selected, and a bit value of 0 indicates the effect is not selected. The effects available are described above.

5.15.4 Function Result

The value returned gives the effects which were selected, in the same form as the parameter `effect`. If a device does not support a particular effect, it will set the corresponding bit to zero in the function result.

5.15.5 Example

```
#define normal      0x0000
#define thickened   0x0001
#define light       0x0002
#define skewed      0x0004
#define underlined  0x0008
#define outlined    0x0010
#define shadowed    0x0020

WORD screen;
WORD effects;
.
.
  effects = vst_effects(screen, shadowed | skewed);
  if (!(effects & skewed))
    { /* skewed style not available */
      ...
    }
```

5.16 Set Graphic Text Alignment `vst_alignment`

When text is output to a device it may be left-aligned, right-aligned or centred relative to a starting point. It may also be placed at various heights above or below the starting point. This function is used to set the horizontal and vertical alignment to be used for subsequent graphic text output.

5.16.1 Definition

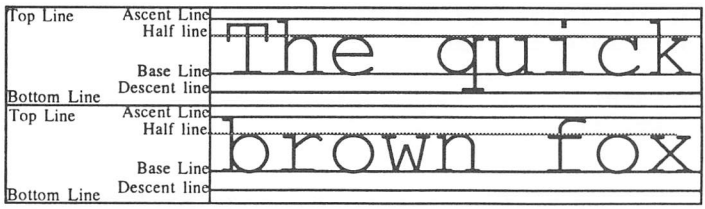
The Prospero C definition of Set Graphic Text Alignment is :

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

5.16.2 Purpose

This function is used to select the alignment of graphic text output. This determines which point of the text output corresponds to the x and y coordinates specified in the parameters passed to the `v_gtext` function (section 4.3). Horizontally, the x coordinate can indicate the start, centre, or end of the text, while vertically the y coordinate may indicate the position of the text base line, half line, ascent line, descent line, cell bottom or cell top (see diagram). The default when a workstation is opened is left base line aligned text.

This function affects the subsequent output of the functions `v_gtext` (section 4.3) and `v_justified` (section 4.14).



Vertical Alignment Points

5.16.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose text alignment is to be set.
hor_in	*WORD*	**Requested horizontal alignment** This parameter specifies the requested horizontal text alignment :- 0 = left aligned (default) 1 = centre aligned 2 = right aligned
vert_in	*WORD*	**Requested vertical alignment** This parameter specifies the requested vertical text alignment :- 0 = base line aligned (default) 1 = half line aligned 2 = ascent line aligned 3 = cell bottom aligned 4 = descent line aligned 5 = cell top aligned
hor_out	*WORD* *	**Selected horizontal alignment** This parameter points to an object which returns the horizontal alignment selected by GEM VDI. If the requested value is out of range, left alignment will be selected.
vert_out	*WORD* *	**Selected vertical alignment** This parameter points to an object which returns the vertical alignment selected by GEM VDI. If the requested value is out of range, base line alignment will be selected.

5.16.4 Example

```
WORD screen_handle;
WORD dummy;
.
.
  /* Select centre cell bottom alignment,
     ignore returned values */
  vst_alignment(screen_handle, 1, 3, &dummy, &dummy);
```

5.17 Set Fill Interior Style `vsf_interior`

GEM VDI can output filled shapes using one of several fill modes – the shapes can be filled with the background color, a specified color, or a pre-defined or user defined pattern. This function is used to select which mode is to be used, and therefore determines which of the other fill attributes are relevant.

5.17.1 Definition

The Prospero C definition of Set Fill Interior Style is :

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

5.17.2 Purpose

This function is used to select the manner in which filled shapes are output by GEM VDI. The possible fill modes are hollow, solid, filled with a pattern or hatch style, or filled with a user defined fill pattern. The initial fill interior style when a workstation is opened is given by the value of `work_in[7]` (see section 3.1).

GEM VDI supports 5 different fill styles, each of which can be further modified using other attribute functions described in later sections :-

0 – hollow fill – the area is filled using the background color

1 – solid fill – the area is filled using the color defined using `vsf_color` (section 5.19)

2 – pattern fill – the area is filled using the pattern whose index is selected using `vsf_style` (section 5.18)

3 – hatch fill – the area is filled using the hatch pattern whose index is selected using `vsf_style` (section 5.18)

4 – user-defined pattern – the area is filled using the pattern defined using `vsf_udpat` (section 5.21)

This function affects the subsequent output of the functions `v_fillarea` (section 4.4), `v_contourfill` (section 4.6), `vr_recfl` (section 4.7), `v_bar` (section 4.8) , `v_pieslice` (section 4.9), `v_circle` (section 4.10), `v_ellpie` (section 4.11), `v_ellipse` (section 4.12), `v_rbox` (section 4.13) and `v_rfbox` (section 4.13).

5.17.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose fill mode is to be set.
style	*WORD*	**Requested fill interior style** This parameter specifies the requested fill interior style, as described above. The value should be in the range 0 to 4.

5.17.4 Function Result

The value returned is the fill mode selected by GEM VDI. Hollow fill (mode zero) is selected if the requested mode is not available or out of range.

5.17.5 Example

```
WORD screen_handle;
.
.
  if (vsf_interior(screen_handle, 3) == 3 )
    /* Hatch mode available so pick one */
    vsf_style(screen_handle, 1);
```

5.18 Set Fill Style Index `vsf_style`

If pattern or hatch fill mode is selected using `vsf_interior` (see section 5.17), this function should be used to select which pattern or hatch style is to be used .

5.18.1 Definition

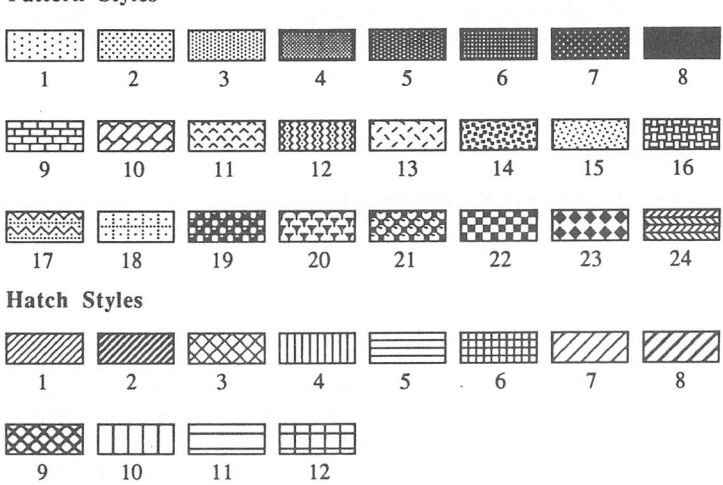
The Prospero C definition of Set Fill Style Index is :

```
WORD vsf_style(WORD handle, WORD style);
```

5.18.2 Purpose

This function is used to select which fill pattern or hatch style will be used for fill operations. The number of patterns and hatches available are returned in `work_out[11]` and `work_out[12]` when the workstation is opened – see section 3.1. Pattern styles 1 to 8 are uniform monochrome dot patterns of increasing intensity, so that fill pattern index 8 is the same as selecting solid fill – see section 5.17. Hatch styles differ from patterns only in that they always consist of horizontal, vertical and diagonal lines in various combinations. The diagram below shows the normal patterns and hatch styles available, though these may vary slightly between devices, and between versions of GEM. The initial fill style index is given by the value of `work_in[8]` when the workstation is opened.

This function affects the subsequent output of the functions `v_fillarea` (section 4.4), `v_contourfill` (section 4.6), `vr_recfl` (section 4.7), `v_bar` (section 4.8), `v_pieslice` (section 4.9), `v_circle` (section 4.10), `v_ellpie` (section 4.11), `v_ellipse` (section 4.12), `v_rbox` (section 4.13) and `v_rfbox` (section 4.13).



GEM VDI Pattern and Hatch Style Indices

5.18.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose fill style index is to be set.
style	*WORD*	**Requested fill style index** This parameter specifies the requested fill style index, in the range 1 to work_out[11] for patterns and 1 to work_out[12] for hatches.

5.18.4 Function Result

The value returned is the fill style index selected by GEM VDI. Index 1 is selected if the requested index is out of range.

5.18.5 Example

```
WORD screen_handle;
int  inc, angle, i;
WORD work_out[57];
.
.
  angle = 0;
  inc = 3600 / (work_out[11] + work_out[12]);

  /* Select pattern fill */
  vsf_interior(screen_handle, 2);

  for (i = 1; i<= work_out[1]; i++)
  {    /* Use each pattern in turn in turn */
    vsf_style(screen_handle, i);
    v_pieslice(screen_handle, 150, 150, 100,
                angle, angle+inc);
    angle += inc;
  };

  /* Select hatch fill */
  vsf_interior(screen_handle, 3);

  for (i = 1; i <= work_out[12]; i++);
  {   /* Use each hatch pattern in turn */
    vsf_style(screen_handle, i);
    v_pieslice(screen_handle, 150, 150, 100,
                angle, angle+inc);
    angle += inc;
  };
```

5.19 Set Fill Color Index `vsf_color`

This function is used to specify the color which is to be used for subsequent fill operations, when the selected fill mode (section 5.17) is solid fill, pattern fill, hatch fill or user defined pattern fill. When hollow fill mode is selected, the area is always filled with the background color.

5.19.1 Definition

The Prospero C definition of Set Fill Color Index is :

```
WORD vsf_color(WORD handle, WORD color);
```

5.19.2 Purpose

This function is used to select the color index used for filling shapes, when using solid fill style, or monochrome (single plane) fill patterns or hatches. Fill patterns with indices in the range 1 to 8 are always monochrome, and will be displayed as increasing intensity dot patterns in the current fill color as defined by this function. User defined patterns may be monochrome single plane patterns, in which case the fill color, background and writing mode will be used to determine the color of each pixel. Multiplane patterns should only be used in replace mode, and are not affected by the fill color index – see section 5.21 for further details.

Available colors range from 0 to a device dependent maximum, returned in `work_out[13]` when the device is opened (see section 3.1) . If the requested color index is out of range, color index 1 will be selected. The initial fill color index is given by the value of `work_in[9]` when the workstation is opened.

This function affects the subsequent output of the functions `v_fillarea` (section 4.4), `v_contourfill` (section 4.6), `vr_recfl` (section 4.7), `v_bar` (section 4.8), `v_pieslice` (section 4.9), `v_circle` (section 4.10), `v_ellpie` (section 4.11), `v_ellipse` (section 4.12), `v_rbox` (section 4.13) and `v_rfbox` (section 4.13).

5.19.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose fill color index is to be set.
color	*WORD*	**Requested fill color index** This parameter specifies the requested fill color index. If the value is out of range, color index 1 is selected.

5.19.4 Function Result

The value returned is the fill color index selected by GEM VDI. Color index 1 is selected if the requested color is out of range.

5.19.5 Example

```
WORD screen_handle;
.
.
  /* Select solid fill in color 5 */
  vsf_interior(screen_handle, 1);
  vsf_color(screen_handle, 5);

  /* Now draw a filled circle */
  v_circle(screen_handle, 150, 150, 100);
```

5.20 Set Fill Perimeter Visibility `vsf_perimeter`

GEM VDI can output filled areas with or without a solid line (in the current fill color) drawn around the area's boundary, and this function is used to specify whether or not the perimeter is to be drawn in this way.

5.20.1 Definition

The Prospero C definition of Set Fill Perimeter Visibility is :

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

5.20.2 Purpose

This function is used to specify whether filled areas should be outlined or not. When a workstation is opened, the default is that outlining is enabled until disabled using this function.

This function affects the subsequent output of the functions `v_fillarea` (section 4.4), `v_contourfill` (section 4.6), `v_bar` (section 4.8), `v_pieslice` (section 4.9), `v_circle` (section 4.10), `v_ellpie` (section 4.11), `v_ellipse` (section 4.12), `v_rbox` (section 4.13) and `v_rfbox` (section 4.13). Note however that `vr_recfl` (section 4.7) does not use this attribute, and never outlines the filled rectangles it outputs. To output an outlined filled rectangle, the function `v_bar` should be used which does use this attribute.

5.20.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose fill perimeter visibility is to be set.
`per_vis`	*WORD*	**Visibility flag** If this is non-zero, the filled area's border will be marked with a solid line. If zero, the border will not be marked.

5.20.4 Example

```
WORD screen_handle;
.
.
  /* Select hollow fill, no border */
  vsf_interior(screen_handle, 0);
  vsf_perimeter(screen_handle, 0);

  /* Erase a circular area */
  v_circle(screen_handle, 150, 150, 100);
```

5.21 Set User Defined Fill Pattern `vsf_udpat`

One of the fill modes which may be selected using `vsf_interior` (section 5.17) is to use the user definable fill pattern. When this mode is in force, the pattern used for filling areas can be defined using this function.

5.21.1 Definition

The Prospero C definition of Set User Defined Fill Pattern is :

```
void vsf_udpat(WORD handle, WORD fill_pat[],
               WORD planes);
```

5.21.2 Purpose

This function is used to define the pattern to be used when the user defined fill interior style is in use. The pattern can be monochrome (one plane), in which case it will be output using the current writing mode, with the foreground color defined by the current fill color index (section 5.19), or it can contain several planes, in which case it must be output in replace mode, any unspecified planes being treated as zero.

The array defining the pattern can thus consist of any number of planes, each of 16 elements. The application should define a suitable array with 16 *WORD* elements per plane, then pass the address of the array to this function in the parameter `fill_pat`. Each element in the array corresponds to a single row of the pattern, with each bit giving the value of a single pixel in that plane. The most significant bit of the first element corresponds to the top left corner of the pattern.

This function affects the subsequent output of the functions `v_fillarea` (section 4.4), `v_contourfill` (section 4.6), `vr_recfl` (section 4.7), `v_bar` (section 4.8), `v_pieslice` (section 4.9), `v_circle` (section 4.10), `v_ellpie` (section 4.11), `v_ellipse` (section 4.12), `v_rbox` (section 4.13) and `v_rfbox` (section 4.13).

5.21.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose user definable fill pattern is to be set.
fill_pat	*WORD []*	**Fill pattern array** The array defining the fill pattern. The application must ensure that the array contains 16 elements for every plane specified by the planes parameter.
planes	*WORD*	**Number of planes** The number of planes in the pattern. A pattern with more than one plane can only be used in replace mode, each plane in the pattern being copied to the corresponding color plane of the device. A pattern with a single plane can be output in any of the four writing modes selected using vswr_mode (section 5.1).

5.21.4 Example

```
WORD screen_handle;
WORD my_pat[64];
WORD other_pat[16];
int  i, j;
.
.
  for (i = 0; i < 4; i++)
    for (j = 0; j < 16; i++)
     /* set up my_pat array */
  for (j = 0; j < 16; j++)
     /* set up other_pat array */

/* Select and define user defined fill, replace mode */
  vsf_interior(screen_handle, 4);
  vsf_udpat(screen_handle, my_pat, 4);
  vswr_mode(screen_handle, 1);

  /* Draw multicolored circle, replace mode */
  v_circle(screen_handle, 150, 150, 100);

  /* Define another pattern, transparent mode */
  vsf_udpat(screen_handle, other_pat, 1);
  vswr_mode(screen_handle, 2);

  /* Draw monochrome pattern , transparent mode */
  v_circle(screen_handle, 250, 150, 100);
```

6 VDI RASTER FUNCTIONS

This section contains descriptions of the VDI Raster Operation functions, in the following sub-sections.

Section	Function description	Binding name
6.1	Copy Raster Opaque	`vro_cpyfm`
6.2	Copy Raster Transparent	`vrt_cpyfm`
6.3	Transform Form	`vr_trnfm`
6.4	Get Pixel	`v_get_pixel`

Raster operations concern the process of copying large amounts data from one area of memory to another, between the device display area and an area of memory, or from one area of the device display to another. One major use for these kind of operations is to move text and/or graphics around the screen, for example to scroll the contents of a window, or to copy an area of the screen to a memory buffer, to be restored at a later date.

6.1 Copy Raster Opaque vro_cpyfm

This function allows rectangular raster areas to be copied from one area of the display to another, or to or from an area of memory. A number of operations can be performed on the raster data between the source and destination, so that this function may be used to clear or invert a portion of the display, although a simple copy is probably the most common use.

6.1.1 Definition

The Prospero C definition of Copy Raster Opaque is :

```
void vro_cpyfm(WORD handle, WORD wr_mode,
               WORD xyarray[8],
               MFDB *srcMFDB, MFDB *desMFDB);
```

6.1.2 Purpose

This function modifies a portion of the memory or display area described by `desMFDB`, depending on the value of the parameter `wr_mode` and the contents of the area described by `srcMFDB`. The two raster areas can be part of the display area or buffer of the device concerned, or they may be blocks of memory. The same memory area may be used for both source and destination, and the portions of the memory area which correspond to the source and destination may overlap. A number of different operations other than simply copying are possible – in fact there are 16 possible modes according to the value of the `wr_mode` parameter as follows :-

```
0  destination = 0
1  destination = source & destination
2  destination = source & (~ destination)
3  destination = source                          /* replace mode */
4  destination = (~ source) & destination        /* erase mode */
5  destination = destination
6  destination = source ^ destination            /* XOR mode */
7  destination = source | destination            /* transparent mode*/
8  destination = ~ (source | destination)
9  destination = ~ (source ^ destination)
10 destination = ~ destination
11 destination = source | (~ destination)
12 destination = ~ source
13 destination = (~ source) | destination        /* reverse transparent */
14 destination = ~ (source & destination)
15 destination = 1
```

The areas of memory used to contain the source and destination rasters are described by records of type *MFDB*. This is declared in the file VDIBIND.H as follows :-

```
typedef struct {
          WORD * praster;
          WORD width, height, word_width;
          WORD standard;
          WORD nplanes;
          WORD res1, res2, res3;  /* Reserved */
               } MFDB;
```

The meanings of the fields are as follows :-

`praster` A pointer to the area of memory to be used to store the raster.

A value of `NULL` indicates that the device display area is to be used rather than an area of memory – this should be used for example to move one area of the screen to another, or to copy the screen to a buffer. If `praster` is `NULL`, the other fields in the record are not relevant.

`width` Specifies the width of the raster area in pixels.

`height` Specifies the height of the raster area in pixels.

`word_width` Specifies the width of the raster area in words. This will be the width in pixels divided by the number of bits in a word (16). This value is used by GEM VDI to calculate the offset in words corresponding to moving down one pixel.

`standard` This is a flag indicating whether the raster is in device specific (`standard` = 0) or device independent standard form (`standard` = 1). For the `vro_cpyfm` function both source and destination must be in device specific form.

`nplanes` The number of planes in the raster area.

6.1.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device concerned.
wr_mode	*WORD*	**Writing mode** This parameter determines how the source and destination raster areas are to interact. There are 16 possible modes as described above.
xyarray	*WORD[8]*	**Rectangle definitions** This parameter contains the coordinates of two rectangles: each point is represented by a consecutive pair of elements; these are the two opposite corners of the source and destination rasters, making a total of four points or eight elements. If both source and destination share the same area of memory (e.g. both refer to the device) and the rectangles overlap, the order in which the copy is performed will be such that no area of the source is altered until it has been processed. If the source and destination rectangles are not the same size, the size of the source will be used, and the location of the destination used unless scaling is available – this can be determined using the function vq_extnd described in section 8.1.

srcMFDB	*MFDB* *	**Source raster definition**
desMFDB	*MFDB* *	**Destination raster definition**

The parameters srcMFDB and desMFDB are memory form definition blocks as described above. To copy from one area of the device to another, a structure of type *MFDB* must be declared, the praster field assigned the value NULL, and the variable passed in both parameters.

6.1.4 Example

```
MFDB src, dest;
WORD rects[] = { 0, 0, 319, 199, /* The whole screen */
                 0, 0, 319, 199  };
WORD screen_handle;
WORD * my_save_area = (WORD *) malloc (32000);
.
.
  src.praster = NULL;    /* No other fields relevant */
  dest.praster = my_save_area;
  dest.width = 320;
  dest.height = 200;
  dest.word_width = 20;      /* 20 words is 320 bits */
  dest.standard = 0;
  dest.nplanes = 4;

  /* Save whole of a four-plane (16 color) screen to
     a memory area, replace mode */
  vro_cpyfm(screen_handle, 3, rects, &src, &dest);
```

6.2 Copy Raster Transparent `vrt_cpyfm`

This function is a little similar to the preceding function Copy Raster Opaque (section 6.1), except that the source raster area is always a monochrome, single plane mask, which must be an area of memory rather than the screen or display buffer. The destination may be the display buffer or a memory area. Four different writing modes are supported to give different interactions between the mask data in the source raster and the destination.

6.2.1 Definition

The Prospero C definition of Copy Raster Transparent is :

```
void vrt_cpyfm(WORD handle, WORD wr_mode,
               WORD xyarray[8],
               MFDB *srcMFDB, MFDB *desMFDB,
               WORD index[2]);
```

6.2.2 Purpose

This function modifies a portion of the raster area described by `desMFDB` depending on the value of the parameter `wr_mode` and the contents of the mask described by `srcMFDB`. The destination raster area can be part of the display area or buffer of the device concerned, or a memory area. The source must be a monochrome (single plane) area of memory. The destination is modified according to the values of the corresponding bits in the source mask, the two color indices provided and the writing mode. Four writing modes are supported as follows :-

1 – (replace mode)
All pixels in the destination rectangle will be set to the foreground color where the source mask has a bit set, and to the background color where the source has a bit not set. The foreground and background colors are specified by `index[0]` and `index[1]` respectively.

2 – (transparent mode)
Those pixels in the destination where the source has a pixel set are set to the foreground color in `index[0]`. Pixels in the destination which correspond to unset pixels in the source are unaffected. The value of `index[1]` is not used.

3 – (XOR mode)
The monochrome raster source is logically XORed with each plane of the destination. The values of `index` are not used.

4 – (reverse transparent mode)
Those pixels in the destination where the source has a bit unset are set to the background color in `index[1]`. Other pixels are not affected. The value of `index[0]` is not used.

These writing modes correspond to those available for normal GEM VDI output, described in section 5.1.

6.2.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device concerned.
`wr_mode`	*WORD*	**Writing mode** This parameter determines how the source mask is to affect the pixels in the destination raster area. There are 4 possible modes as described above.
`xyarray`	*WORD[8]*	**Rectangle definitions** This parameter contains the co-ordinates of two rectangles: each point is represented by a consecutive pair of elements; these are the two opposite corners of the source and destination rasters, making a total of four points or eight elements. If the source and destination rectangles are not the same size, the size of the source will be used, and the location of the top left hand corner of the destination used to determine which area of the destination is affected.

`srcMFDB` *MFDB ** **Source raster definition**
`desMFDB` *MFDB ** **Destination raster definition**

The parameters `srcMFDB` and `desMFDB` point to memory form definition blocks, as described under the Copy Raster Opaque function in section 6.1. The source must be a memory form rather than the device, and must consist of a single plane.

`index` *WORD[2]* **Color indices**

This parameter is an array of two two-byte integers, giving the foreground and background colors respectively. The manner in which they affect the destination color depends upon the value of the parameter `wr_mode`, as described above.

6.2.4 Example

```
WORD mask[16];
MFDB src, dest;
WORD index[2];
WORD myrects[8];
.
.
  /* First set up a shape in the array mask */
  src.praster = &mask;
  src.width = 16;          /* 16 bits in a word */
  src.height = 16;
  src.word_width = 1;
  src.standard = 0;
  src.nplanes = 1;         /* Mask must be monochrome */

  dest.praster = NULL;     /* Destination is the screen */
  myrects[0] = 0;          /* Set up source rectangle */
  myrects[1] = 0;
  myrects[2] = 15;
  myrects[3] = 15;         /* whole of source area */
  index[0] = 3;            /* new color where mask set */

  /* Set up myrects[4 to 7] for required dest area */

  /* Set destination to color 3 where mask was 1 */
  vrt_cpyfm(screen, 2, myrects, &src, &dest, index);
```

6.3 Transform Form `vr_trnfm`

This function is used to transform a raster form from device specific to standard form, or vice versa. A program might convert a raster area to standard form then to the device specific form of a different device, to transfer a raster from one device to another.

6.3.1 Definition

The Prospero C definition of Transform Form is :

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

6.3.2 Purpose

This function places in the area described by the destination Memory Form Definition Block (MFDB) a copy of the raster form described by the source MFDB, except that if the source is in device specific form the destination will be in standard form, and vice versa. The number of planes transferred is specified by the source MFDB. The format flag in the source MFDB (the value of the `standard` field) is toggled and placed in the destination MFDB – other fields in the destination MFDB must be set up by the application. The source and destination may be the same area, otherwise they must not overlap in memory. The physical device, specified by the value `NULL` in the `praster` field of a MFDB, can only be in device specific form. See section 6.1 for a description of the fields of an MFDB.

In standard form, the mapping between colors and pixel values is fixed, and the planes are stored as contiguous rows of words from the top left of the image, with the most significant bit of each 16-bit word corresponding to the leftmost bit in the image.

6.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device concerned.
srcMFDB	*MFDB **	**Source raster definition**
desMFDB	*MFDB **	**Destination raster definition** The parameters srcMFDB and desMFDB point to memory form definition blocks, as described under the Copy Raster Opaque function in section 6.1. The number of planes specified by srcMFDB.nplanes will be transferred to the destination, and the form (standard or device specific) of the source will be toggled and placed in the destination MFDB.

6.3.4 Example

```
MFDB src;
WORD screen_handle;
.
.
  /* Copy a raster area into memory form described by
     src   (e.g. using vro_cpyfm) */

  /* Transform it to standard form */
  vr_trnfm(screen_handle, &src, &src);
```

6.4 Get Pixel v_get_pixel

This function is used return the pixel value and color index for a specified pixel. The pixel value is the combined bit values of each plane of the display at the specified point, while the color index is the value which, when used as the line or fill color, for example, would produce that pixel value. The mapping between pixel values and color indices is device specific.

6.4.1 Definition

The Prospero C definition of Get Pixel is :

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

6.4.2 Purpose

This function is used to determine the pixel value and color index of a specified pixel element of the device.

6.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose pixels are to be tested.
x y	*WORD* *WORD*	**X coordinate** **Y coordinate** The x and y coordinates of the point in question in the current coordinate system. The pixel containing the specified point will be tested.
pel	*WORD **	**Pixel value** This parameter points to an object which returns the pixel value of the specified point, in device specific form.
index	*WORD **	**Color index** This parameter points to an object which returns the color index of the specified pixel. Color index 0 is the background color, but may not be represented by pixel value 0 in the device specific form.

6.4.4 Example

```
WORD screen_handle;
WORD value, color;
.
.
  /* Get pixel value and color of point (100,100) */
  v_get_pixel(screen_handle, 100, 100,
              &value, &color);
```

7 VDI INPUT FUNCTIONS

This section contains detailed descriptions of the GEM VDI input functions, in the following sub-sections.

Section	Function description	Binding name
7.1	Set Input Mode	`vsin_mode`
7.2	Input Locator	`vrq_locator` `vsm_locator`
7.3	Input Valuator	`vrq_valuator` `vsm_valuator`
7.4	Input Choice	`vrq_choice` `vsm_choice`
7.5	Input String	`vrq_string` `vsm_string`
7.6	Set Mouse Form	`vsc_form`
7.7	Exchange Timer Vector	`vex_timv`
7.8	Show and Hide Cursor	`v_show_c` `v_hide_c`
7.9	Sample Mouse State	`vq_mouse`
7.10	Exchange Button Change Vector	`vex_butv`
7.11	Exchange Mouse Travel Vector	`vex_motv`
7.12	Exchange Cursor Draw Vector	`vex_curv`
7.13	Sample Keyboard State	`vq_key_s`

The routines described in this section are concerned with obtaining input from the user. GEM VDI input uses the concept of logical input devices associated with each workstation, so that a program can obtain input from the locator, the valuator, the choice or the string logical device according to what information the program requires. These logical devices are described further under the relevant sub-section.

Each logical device can operate in sample mode or request mode, as described in section 7.1, and a separate function or function is provided in each mode. These functions differ only in the way that the parameters are passed to the GEM VDI, but make use of the same GEM VDI function. Thus calling the request mode function when a device is in sample mode will not cause sample mode to be entered, but will cause the values to be passed to and from GEM VDI in a manner which may not be relevant in sample mode. Therefore a program should always take care that the function or function used corresponds to the current state of the logical device. When a workstation is opened, all devices are initially in request mode.

Programs which use the AES must use the AES input functions rather than the ones in this section. Many of the routines in this section have direct counterparts in the AES which perform identical functions.

7.1 Set Input Mode `vsin_mode`

GEM VDI supports input from a number of logical devices, each of which may operate in one of two modes – sample mode or request mode. In request mode, the program calls a function whenever it wants input from that logical device, and the function returns when the required input is available. In sample mode, the program will call the relevant function when it requires input, but the function will return immediately whether or not input was available. In this way the application could, for example, continue processing while waiting for the input. Note that each logical input device has two input functions to be used with it, one for sample mode and one for request mode. However these functions do not cause the required input mode to be selected, so that a program must take care to ensure it always sets the correct input mode using this function before requesting input.

When several virtual screen workstations share the same physical screen, sample mode input cannot operate correctly, as GEM VDI cannot tell for which workstation an input was intended. Request mode only should be used in such cases.

7.1.1 Definition

The Prospero C definition of Set Input Mode is:

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

7.1.2 Purpose

This function allows the input mode for a particular logical input device to be selected for subsequent input. Each of the (up to) four logical input devices that a device may support operates in two modes, request mode and sample mode, according to whether the device is to wait until input is available or return immediately indicating whether input was available or not.

Two input functions are provided for each logical input device, one for sample mode input and one for request mode input. However, it is important to realize that these are simply different interfaces to the same GEM function, and that these functions do not cause the device to enter request or sample mode. Therefore the desired input mode should be set using the function `vsin_mode` before using the input functions, to place the input device in the correct input mode. When a workstation is opened, all input devices will be placed in request mode.

7.1.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose input mode is to be set.
dev_type	*WORD*	**Input device type** The logical input device whose input mode is being selected, as follows :- 1 : locator 2 : valuator 3 : choice 4 : string
mode	*WORD*	**Input mode** The input mode to be used for subsequent input from the logical device :- 1 – Request mode 2 – Sample mode Note that sample mode should not be used when multiple virtual screen workstations are sharing the same physical screen device, as the driver cannot tell which virtual workstation should receive the input, and therefore input may be missed.

7.1.4 Example

```
WORD screen_handle;
.
.
  /* Open physical screen workstation */
  /* Select sample mode for string input device */
  vsin_mode(screen_handle, 4, 2);
```

7.2 Input Locator

`vrq_locator`
`vsm_locator`

These functions are used to obtain input from the locator logical input device. The function used depends upon the currently selected input mode for the locator device (see `vsin_mode` in section 7.1) – use `vrq_locator` in request mode, or `vsm_locator` in sample mode.

7.2.1 Definition

The Prospero C definitions of Input Locator are:

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

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

7.2.2 Purpose

These are used to obtain a coordinate pair from the locator logical input device associated with a workstation. The locator device may be driven by moving an attached mouse, light pen or pen on a graphics tablet – most screen devices also support the use of the cursor keys for moving the locator device, typically causing large increments when the cursor keys are used unmodified, or small increments when the shift key is held down at the same time. Where the locator device can be operated by an external mouse etc. and the keyboard, either will be tracked on the screen. The value returned in `work_out[40]` when the workstation is opened indicates what capabilities are available – see section 3.1.

The function `vrq_locator` operates in request mode (the default when a workstation is opened), and causes a cursor to be displayed at the starting location and tracked on the screen until a terminating event such as a keypress or a mouse button occurs. The function `vsm_locator` operates in sample mode, which should have been previously selected using `vsin_mode` (see section 7.1). This function does not cause a cursor to be displayed – this can be done via `v_show_c` described in section 7.8.

7.2.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose locator device is to be sampled.
initx inity	*WORD* *WORD*	**Initial X coordinate** **Initial Y coordinate** The initial x and y coordinates of the locator device, in the current coordinate system.
xout yout	*WORD* * *WORD* *	**Final X coordinate** **Final Y coordinate** These parameters point to objects used to return the final x and y coordinates of the locator device, in the current coordinate system.
term	*WORD* *	**Terminating key** This parameter points to an object which returns the ASCII code of the key which was pressed to terminate the input. For locator devices other than the keyboard, terminating events will be returned as values starting at the ASCII space character.

7.2.4 Function Result

The function `vsm_locator` returns a value in the range 0 to 3 as follows :-

0 – Nothing happened
1 – Coordinates changed, no terminating key pressed
2 – Terminating key pressed (returned in `term`), coordinates unchanged
3 – Terminating key pressed (returned in `term`), coordinates changed

7.2.5 Example

```
WORD screen;
WORD xcoord, ycoord;
WORD dummy;             /* Don't care how it ends */
.
.
  /* Open physical screen workstation */

  /* Get a location */
  vrq_locator(screen, 100, 100, &xcoord, &ycoord,
                                              &dummy);
  .
  .
  /* Select sample mode for locator input device */
  vsin_mode(screen, 1, 2);
  v_show_c(screen, 0);        /* Display cursor */

  do {
    /* Sample until terminated */
  } while (vsm_locator(screen, xcoord, ycoord,
                    &xcoord, &ycoord, &dummy)  < 2);
  v_hide_c(screen);
```

7.3 Input Valuator

vrq_valuator
vsm_valuator

These functions are used to obtain input from the valuator logical input device. The function used depends upon the currently selected input mode for the valuator device (see `vsin_mode` in section 7.1) – use `vrq_valuator` in request mode, or `vsm_valuator` in sample mode.

7.3.1 Definition

The Prospero C definitions of Input Valuator are:

```
void vrq_valuator(WORD handle, WORD val_in,
                  WORD *val_out, WORD *term);

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

7.3.2 Purpose

These are used to obtain a value from the valuator logical input device associated with a workstation. The valuator device may be driven by an attached potentiometer, but can also be driven using the up and down cursor keys, typically adding or subtracting ten when the cursor keys are used unmodified, or one when the shift key is held down at the same time. Valuator numbers range from 1 to 100. The value returned in `work_out[41]` when a workstation is opened indicates whether a valuator device is available – see section 3.1.

In request mode, the function `vrq_locator` is used, which will keep adjusting the valuator according to cursor key presses until a terminating key is pressed. In sample mode, `vsm_locator` should be used – this will return immediately indicating whether any change was made to the valuator device, and whether a terminating character was pressed.

These functions are not required and may not be supported by all devices.

7.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose valuator is to be sampled.
val_in	*WORD*	**Initial valuator value** The initial value to be given to the valuator device, in the range 1 to 100.
val_out	*WORD* *	**Final valuator value** Points to an object used to return the final valuator value when the function returns.
term	*WORD* *	**Terminating key** This parameter points to an object which returns the ASCII code of the key which was pressed to terminate the input.
status	*WORD* *	**Status** This parameter is for vsm_valuator only, and points to an object which returns a number in the range 0 to 2 as follows :- 0 – Nothing happened 1 – Valuator changed 2 – Terminator character pressed

7.3.4 Example

```
WORD screen;
WORD result, status;
WORD dummy;              /* Don't care how it ends */
.
.
  /* Open physical screen workstation */

  /* Get a value */
  vrq_valuator(screen, 50, &result, &dummy);
  .
  .
  /* Select sample mode for valuator input device */
  vsin_mode(screen, 2, 2);
  do {
    /* Sample until terminated */
    vsm_valuator(screen, result,
                 &result, &dummy, &status);
  } while ( status != 2);
```

7.4 Input Choice

vrq_choice
vsm_choice

These functions are used to obtain input from the choice logical input device. The function used depends upon the currently selected input mode for the choice device (see vsin_mode in section 7.1) – use vrq_choice in request mode, or vsm_choice in sample mode.

7.4.1 Definition

The Prospero C definitions of Input Choice are :

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

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

7.4.2 Purpose

These are used to obtain a choice value from the choice logical input device of the specified workstation. The choice device is typically driven by the function keys of the keyboard. In request mode, the function vrq_choice is used, which will wait until a key is pressed before returning, and return that choice value if it is a valid choice key, otherwise returning the initial choice value. In sample mode, vsm_choice should be used – this will return immediately indicating whether any choice key has been pressed. The value returned in work_out[42] when a workstation is opened indicates whether a choice device is available – see section 3.1.

These functions are not required and may not be supported by all devices.

7.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose choice input is to be sampled.

`ch_in`	*WORD*	**Default choice value** The default choice value, which will be returned via `ch_out` if the key pressed to terminate the input is not a valid choice key. (`vrq_choice` only)
`ch_out`	*WORD **	**Output choice value** Points to an object used to return the final choice value when the function returns. (`vrq_choice` only)
`choice`	*WORD **	**Choice value** Points to an object used to return the choice if a valid choice was made, otherwise returns 0. (`vsm_choice` only)

7.4.4 Function Result

The function `vsm_choice` returns non-zero if the sample was successful, and the value returned in the parameter `choice` is a valid choice number.

7.4.5 Example

```
WORD screen;
WORD result;
.
.
  /* Open physical screen workstation */
  vrq_choice(screen, 5, &result);     /* Get a choice */
  .
  .
  /* Select sample mode for choice input device */
  vsin_mode(screen, 3, 2);
  do {
    /* Sample until successful */
  } while (!vsm_choice(screen, &result));
```

7.5 Input String

vrq_string
vsm_string

These functions are used to obtain input from the string logical input device. The function used depends upon the currently selected input mode for the string device (see vsin_mode in section 7.1) – use vrq_string in request mode, or vsm_string in sample mode.

7.5.1 Definition

The Prospero C definitions of Input String are :

```
void vrq_string(WORD handle, WORD length,
                WORD echo_mode, WORD echo_xy[2],
                char astring[]);

WORD vsm_string(WORD handle, WORD length,
                WORD echo_mode, WORD echo_xy[2],
                char astring[]);
```

7.5.2 Purpose

These are used to obtain a string value from the string logical input device (the keyboard) of the specified workstation. In request mode, vrq_string is used, and input characters are accumulated until either a carriage return is pressed or the maximum string length requested is reached. In sample mode, vsm_string is used, and characters are accumulated as before, but the function returns as soon as characters are not available at the keyboard as well as either of the other two terminating conditions. The value returned in work_out[43] when a workstation is opened indicates whether a string device is available – see section 3.1.

The string may optionally be echoed at a given location, using the current text attributes (see section 5), but this facility may not be available on all devices.

7.5.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device from which a string is to be input.
length	*WORD*	**Maximum string length** The maximum number of characters to be accumulated before returning. This should not be greater than the length of the receiving character array.
echo_mode	*WORD*	**Echo mode flag** If this is non-zero, the characters typed will be echoed at the given location, using the current text attributes to output the characters. If this is zero, no output will occur. Not all devices support echo mode (in fact most seem not to).
echo_xy	*WORD[2]*	**Echo location** This parameter provides the alignment point (see vst_alignment in section 5) at which the text is to be echoed if echo_mode is non-zero.
astring	*char[]*	**String input** This parameter specifies the character array used to return the string input.

7.5.4 Function Result

The function vsm_string returns the number of characters returned in the parameter astring, or zero if the sample was not successful.

7.5.5 Example

```
WORD screen;
WORD echo_point[2] = { 100, 100};
char result[81];
.
.
  /* Open physical screen workstation */
  /* Get a string */
  vrq_string(screen, 80, 1, echo_point, result);
  .
  .
  /* Select sample mode for string input device */
  vsin_mode(screen, 4, 2);
  if (vsm_string(screen, 1, 0, echo_point, result))
    {  /* A string is available */
    .
    .
```

7.6 Set Mouse Form `vsc_form`

This function may be used to redefine the cursor form used for locator input or at other times when the cursor is displayed, for example by the use of `v_show_c` described in section 7.8. This function should not be used by applications using the GEM AES library – a similar function is available using `graf_mouse` described in section 8 of the AES manual.

7.6.1 Definition

The Prospero C definition of Set Mouse Form is :

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

7.6.2 Purpose

This function can be used to alter the cursor form for VDI applications. The cursor is described by an array[37] of *WORD* as follows :-

`cur_form[0]`	Hotspot x coordinate
`cur_form[1]`	Hotspot y coordinate
`cur_form[2]`	Reserved, must be 1
`cur_form[3]`	Mask color
`cur_form[4]`	Data color
`cur_form[5]` to `cur_form[20]`	Mask
`cur_form[21]` to `cur_form[36]`	Data

The hot spot is the point which is returned or provided as the mouse position – for an arrow cursor this would be the point of the arrow, while for a cross hair type cursor form it would be the centre of the cross. Elements 0 and 1 give the offsets of the hot spot from the top left hand corner of the image.

The image of the mouse form is stored as two 16 by 16 arrays of bits, where the most significant bit of the first element corresponds to the top left hand corner of the mouse image. The first array contains the mask – a one bit in this image causes the corresponding pixel to be set to the mask color, unless the corresponding bit in the data image is also set. The second array contains the data image – a one bit in this image causes the corresponding pixel to be set to the data color.

Normally the mask color will be the background color, and the data color the foreground color, and the mask image will be the same shape as the data image but with a border one pixel wide around it. This produces a cursor which will be visible everywhere on the screen, and will work on every screen. However, varying these 'rules' can produce interesting effects.

7.6.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose mouse form is to be set.
cur_form	*WORD[37]*	**Mouse form** This is an array describing the required mouse form, as described above.

7.6.4 Example

```
WORD screen;
WORD my_cursor[37] = {8, 8, /* Hotspot near centre */
                      1,
                      0,    /* Mask color index 0 */
                      4,    /* Data color index 4 */
                      0x1c, 0x1c, 0x1c, 0x1c,
                      0x1c, 0x1c, 0x1c, 0x1c,
                      0x1c, 0x1c, 0x1c, 0x1c,
                      0x1c, 0x1c, 0x1c, 0x1c,
                      /* Wide vertical bar for mask */
                      0x08, 0x08, 0x08, 0x08,
                      0x08, 0x08, 0x08, 0x08,
                      0x08, 0x08, 0x08, 0x08,
                      0x08, 0x08, 0x08, 0x08
                      /* Thin vertical bar for data */
                     };

  vsc_form(screen, my_cursor);  /* Use my mouse form */
```

7.7 Exchange Timer Vector `vex_timv`

This function may be used to cause an assembler coded routine to be entered every time a timer tick occurs.

7.7.1 Definition

The Prospero C definition of Exchange Timer Vector is :

```
void vex_timv(WORD handle, ENTRY_TYPE tim_addr,
              ENTRY_TYPE *otim_addr, WORD *tim_conv);
```

7.7.2 Purpose

This function can be used to provide the address of an assembler routine which will be called by GEM VDI every time a timer tick occurs.

All registers must be saved, so the code would have to be in assembler rather than Prospero C. The function returns the old value of the timer vector via the parameter `otim_addr`, so that this can be restored when the application has completed. The parameter `tim_conv` is used to return the number of milliseconds per timer tick, and hence the interval which will elapse between successive calls of the application's code.

For 68000 family processors, the application's code is invoked with interrupts disabled, by a JSR instruction. The code should execute without enabling interrupts, and return with a RTS instruction when done.

For 8086 type processors, the application's code is invoked with interrupts disabled, by a FAR CALL instruction. The code should execute without enabling interrupts, and return with a FAR RET instruction when done.

7.7.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose timer vector is to be set.
tim_addr	*ENTRY_ TYPE*	**Timer routine address** This is the address of the routine which GEM VDI will call once per timer tick.
otim_addr	*ENTRY_ TYPE* *	**Old timer routine address** This points to an object which will receive the address of the routine which GEM VDI was previously calling once per timer tick. This should be saved by the application to be used in a subsequent call of vex_timv to restore normal timer operation before terminating.
tim_conv	*WORD* *	**Milliseconds per tick** This points to an object which will receive the number of milliseconds which will elapse between subsequent calls of the application's timer routine.

7.7.4 Example

```
WORD screen;
ENTRY_TYPE save_timer, my_timer;
WORD millisecs;

/* Get address of assembler routine in my_timer first*/
...
/* Cause it to be executed once per tick */
vex_timv(screen, my_timer, &save_timer, &millisecs);
...
/* About to terminate - restore normal timer routine */
vex_timv(screen, save_timer, &save_timer, &millisecs);
/* NB Last two parameters not relevant here! */
```

7.8 Show and Hide Cursor

`v_show_c`
`v_hide_c`

These functions may be used to cause the mouse cursor to be displayed or hidden.

7.8.1 Definition

The Prospero C definitions of Show Cursor and Hide Cursor are :

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

void v_hide_c(WORD handle);
```

7.8.2 Purpose

These two functions show or hide the cursor respectively. Note that two consecutive hides will normally require two shows to restore the cursor – however if the parameter `reset` is non-zero, a call of `v_show_c` will cause the cursor to be displayed however many times `v_hide_c` has been called. A program can thus hide the cursor at the start of a drawing routine, then by using `v_show_c` with a value of zero, restore the cursor only if it was displayed previously, so that the routine has no permanent effect on the mouse state.

The default state when the workstation is opened is that the cursor is hidden, requiring a single call of `v_show_c` to display it.

7.8.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose mouse form is to be displayed or hidden.
reset	*WORD*	**Reset flag** If this is non-zero, the cursor will be displayed regardless of the number of preceding v_hide_c calls that have been made. If reset is zero, then the cursor will be displayed only when the number of v_show_c calls made is equal to the number of v_hide_c calls made. (v_show_c only)

7.8.4 Example

```
WORD screen;
.
.
  /* Remove cursor if onscreen */
  v_hide_c(screen);

  /* Do some drawing */

  /* Replace it if it was onscreen */
  v_show_c(screen, 0);
```

7.9 Sample Mouse State `vq_mouse`

This function may be used to discover the current mouse button states, and the position of the mouse cursor.

7.9.1 Definition

The Prospero C definition of Sample Mouse State is :

```
void vq_mouse(WORD handle, WORD *status,
              WORD *x, WORD *y);
```

7.9.2 Purpose

This function is used to return the state (up or down) of the mouse buttons, and the current mouse location. Note that applications which use the GEM AES should use the `graf_mkstate` routine described in the AES manual section 8, which performs a similar function.

7.9.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose mouse state is to be returned.
status	*WORD* *	**Button status** This parameter points to an object which returns a map of which of the (up to) 16 mouse buttons are depressed. The least significant bit corresponds to the leftmost button, with a bit value of 1 indicating the corresponding mouse button is depressed. Thus for example with a two button mouse, the following values may be returned :- 0 (00 binary) – No buttons down 1 (01 binary) – Left hand button down 2 (10 binary) – Right hand button down 3 (11 binary) – Both buttons down
x y	*WORD* * *WORD* *	**Mouse X coordinate** **Mouse Y coordinate** These parameters point to objects which return the current location of the mouse cursor, in the current coordinate system.

7.9.4 Example

```
WORD screen;
WORD state;
WORD x, y;
.
.
  do {
    /* Get mouse state */
    vq_mouse(screen, &state, &x, &y);
  } while ( state != 1);    /* Wait until left hand
                               button down */
```

7.10 Exchange Button Change Vector `vex_butv`

This function may be used to cause an assembler coded routine to be entered every time the mouse button state changes.

7.10.1 Definition

The Prospero C definition of Exchange Button Change Vector is :

```
void vex_butv(WORD handle, ENTRY_TYPE usrcode,
              ENTRY_TYPE *savcode);
```

7.10.2 Purpose

This function can be used to provide the address of an assembler routine which will be called by GEM VDI every time it notices a change in the mouse button state. This routine might for example be used to translate button combinations, so that a mouse with 4 buttons could be made to work as two identical button pairs.

All registers must be saved, so the code would have to be in assembler rather than Prospero C. The function returns the old value of the button change vector via the parameter `savcode`, so that this can be restored when the application has completed, or when normal button function is to be restored.

For 68000 family processors, the application's code is invoked with interrupts disabled, by a JSR instruction. The code should execute without enabling interrupts, and return with a RTS instruction when done. The register D0.w contains the new mouse button state – this may be modified by the routine to indicate what button state is to be saved by the driver. All other registers must be preserved.

For 8086 family processors, the application's code is invoked with interrupts disabled, by a FAR CALL instruction. The code should execute without enabling interrupts, and return with a FAR RET instruction when done. The register AX contains the new mouse button state – this may be modified by the routine to indicate what button state is to be saved by the driver. All other registers must be preserved.

7.10.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose button change vector is to be set.
usrcode	*ENTRY_TYPE*	**Button change routine address** This is the address which GEM VDI will call when it detects a change in the button state.
savcode	*ENTRY_TYPE* *	**Old button change routine** This points to an object which receives the address of the routine which GEM VDI was previously calling when the button state changed. This should be saved by the application to be used in a subsequent call of `vex_butv` to restore normal button operation before terminating.

7.10.4 Example

```
WORD screen;
ENTRY_TYPE save_button, my_button;
.
.
  /* Get address of assembler routine in my_button */

  /* Cause it to be called when button state changes */
  vex_butv(screen, my_button, &save_button);
  .
  .
  /* About to terminate - restore old button routine */
  vex_butv(screen, save_button, &save_button);
  /* Last parameter not relevant here! */
```

7.11 Exchange Mouse Travel Vector `vex_motv`

This function may be used to cause an assembler coded routine to be entered every time the mouse moves to a new location.

7.11.1 Definition

The Prospero C definition of Exchange Mouse Travel Vector is :

```
void vex_motv(WORD handle, ENTRY_TYPE usrcode,
              ENTRY_TYPE *savcode);
```

7.11.2 Purpose

This function can be used to provide the address of an assembler routine which will be called by GEM VDI every time it notices a mouse movement. This routine might for example be used to scale mouse movements, for example to make a large mouse movement have only a small effect.

All registers must be saved, so the code would have to be in assembler rather than Prospero C. The function returns the old value of the mouse travel vector in the parameter `savcode`, so that this can be restored when the application has completed, or when normal mouse function is to be restored.

For 68000 family processors, the application's code is invoked with interrupts disabled, by a JSR instruction. The code should execute without enabling interrupts, and return with a RTS instruction when done. The registers D0.w and D1.w contain the new mouse position – these may be modified by the routine to indicate what position is to be saved by the driver. All other registers must be preserved.

For 8086 type processors, the application's code is invoked with interrupts disabled, by a FAR CALL instruction. The code should execute without enabling interrupts, and return with a FAR RET instruction when done. The registers BX and CX contain the new mouse position – these may be modified by the routine to indicate what position is to be saved by the driver. All other registers must be preserved.

7.11.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose mouse travel vector is to be set.
usrcode	*ENTRY_TYPE*	**Mouse travel routine address** This is the address which GEM VDI will call when it detects a mouse travel.
savcode	*ENTRY_TYPE **	**Old mouse travel routine** This points to an object which receives the address of the routine which GEM VDI was previously calling when the mouse moved. This should be saved by the application to be used in a subsequent call of vex_motv to restore normal mouse operation before terminating.

7.11.4 Example

```
WORD screen;
ENTRY_TYPE save_mouse, my_mouse;
.
.
  /* Get address of assembler routine in my_mouse */
  ...
  /* Cause it to be entered whenever mouse moves */
  vex_motv(screen, my_mouse, &save_mouse);
  .
  .
/* About to terminate - restore normal mouse routine */
  vex_motv(screen, save_mouse, &save_mouse);
  /* Last parameter not relevant here! */
```

7.12 Exchange Cursor Draw Vector `vex_curv`

This function may be used to cause an assembler coded routine to be entered every time GEM VDI is about to redraw the cursor.

7.12.1 Definition

The Prospero C definition of Exchange Cursor Draw Vector is :

```
void vex_curv(WORD handle, ENTRY_TYPE usrcode,
              ENTRY_TYPE *savcode);
```

7.12.2 Purpose

This function can be used to provide the address of an assembler routine which will be called by GEM VDI every time it is about to redraw the mouse cursor. This routine might for example be used to draw a cursor form which does not conform to the two-color, 16 by 16 pixel image used by GEM VDI.

All registers must be saved, so the code would have to be in assembler rather than Prospero C. The function returns the old value of the cursor draw vector via the parameter `savcode`, so that this can be restored when the application has completed, or when normal cursor drawing function is to be restored.

For 68000 family processors, the application's code is invoked with interrupts disabled, by a JSR instruction. The code should execute without enabling interrupts, and return with a RTS instruction when done. The registers D0.w and D1.w contain the x and y locations at which the cursor is to be drawn. If the routine does not cause a cursor to be drawn, it must perform a JSR to the address returned via the parameter `savcode` with D0.w and D1.w containing the position at which the cursor is to be drawn. All other registers must be preserved.

For 8086 family processors, the application's code is invoked with interrupts disabled, by a FAR CALL instruction. The code should execute without enabling interrupts, and return with a FAR RET instruction when done. The registers BX and CX contain the x and y locations at which the cursor is to be drawn. If the routine does not cause a cursor to be drawn, it must perform a CALL (FAR) to the address returned via the parameter `savcode` with BX and CX containing the position at which the cursor is to be drawn. All other registers must be preserved.

7.12.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device whose cursor draw vector is to be set.
usrcode	*ENTRY_TYPE*	**Cursor draw routine address** This is the address of the routine which GEM VDI will call when it is about to draw the cursor.
savcode	*ENTRY_TYPE* *	**Old cursor draw routine** This points to an object which will receive the address of the routine which GEM VDI was previously calling to draw the cursor. This should be saved by the application to be used in a subsequent call of vex_curv to restore normal cursor operation before terminating.

7.12.4 Example

```
WORD screen;
ENTRY_TYPE save_cursor, my_cursor;
.
.
/* Get address of assembler routine in my_cursor */
  ...
/* Cause it to be called when cursor is to be drawn */
  vex_curv(screen, my_cursor, &save_cursor);
  .
  .
/* About to terminate - restore old cursor routine */
  vex_curv(screen, save_cursor, &save_cursor);
  /* Last parameter not relevant here! */
```

7.13 Sample Keyboard State `vq_key_s`

This function may be used to discover the current state of the shift, control, and ALT keys.

7.13.1 Definition

The Prospero C definition of Sample Keyboard State is :

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

7.13.2 Purpose

This function is used to return the state (up or down) of the shift, control and ALT keys, so that they can for example be used to modify the effect of mouse actions. Note that applications which use the GEM AES should use the `graf_mkstate` routine described in the AES manual section 8, which performs a similar function.

7.13.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose keyboard state is to be returned.
`status`	*WORD **	**Key status** This parameter points to an object which receives a map of which of the relevant keys are depressed. Each bit corresponds to a particular key, with a bit value of 1 indicating the corresponding key is depressed. The bits are assigned as follows :- 0 (0000 binary) – No buttons down 1 (0001 binary) – Right hand shift down 2 (0010 binary) – Left hand shift down 4 (0100 binary) – Control key down 8 (1000 binary) – ALT key down These values may be combined to indicate that a combination of the above keys are pressed.

7.13.4 Example

```
WORD screen;
WORD state;
.
.
  do {
    /* Get keyboard state */
    vq_key_s(screen, &state);
  } while ( state != 4);  /* Wait until ctrl key
                             pressed */
```

8 VDI INQUIRE FUNCTIONS

This section contains detailed descriptions of the VDI Inquire functions, in the following sub-sections.

Section	Function description	Binding name
8.1	Extended Inquire	`vq_extnd`
8.2	Inquire Color Representation	`vq_color`
8.3	Inquire Line Attributes	`vql_attributes`
8.4	Inquire Marker Attributes	`vqm_attributes`
8.5	Inquire Fill Attributes	`vqf_attributes`
8.6	Inquire Text Attributes	`vqt_attributes`
8.7	Inquire Text Extent	`vqt_extent`
8.8	Inquire Character Cell Width	`vqt_width`
8.9	Inquire Font Name and Index	`vqt_name`
8.10	Inquire Cell Array	`vq_cellarray`
8.11	Inquire Input Mode	`vqin_mode`
8.12	Inquire Font Info	`vqt_font_info`
8.13	Inquire Justified Graphic Text	`vqt_justified`

The functions described in this section are concerned with obtaining information about the current settings of GEM VDI attributes which affect the output produced using the functions described in section 4, and other information which might be of use to a program which needs to know precisely how its output will appear. Most of the functions in this section test values which can be altered using the attribute functions described in section 5 – further details of the meanings of these values can be found under the corresponding functions in that section.

8.1 Extended Inquire `vq_extnd`

This can be used to obtain further information about a workstation, in addition to that returned by the function `v_opnwk` or `v_opnvwk` when the workstation was opened.

8.1.1 Definition

The Prospero C definition of Extended Inquire is:

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

8.1.2 Purpose

This function is used to return a number of additional items of information about the specified workstation. It may also be used to return the same information as returned by `v_opnwk` or `v_opnvwk` when the workstation was opened, which might be useful when for example the program currently running did not open the workstation itself, but 'inherited' it from a parent program.

8.1.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
`einqflag`	*WORD*	**Extended inquire flag** If the value of this parameter is 0, the information returned is exactly the same as that returned by the `v_opnwk` or `v_opnvwk` functions – see section 3.1. However, if the value of this parameter is non-zero, a new set of information is returned as described below.

`work_out` *WORD[57]* **Workstation information**

This is an array[57] of *WORD*. If the parameter `einqflag` has the value 0, the values assigned to `work_out` are as described under `v_opnwk` in section 3.1. If `einqflag` is non-zero, the information returned is as follows :-

`work_out[0]` Type of screen
- 0 Not a screen device
- 1 Separate alpha and graphic controllers and screens
- 2 Separate alpha and graphic controllers with a common screen
- 3 Common alpha and graphic controllers with separate image memory
- 4 Common alpha and graphic controllers and image memory

`work_out[1]` Number of background colors available in color palette

`work_out[2]` Text effects supported (see `vst_effects` in section 5.15)

`work_out[3]` Raster scaling supported (0 = no, 1 = yes)

`work_out[4]` Number of planes

`work_out[5]` Lookup table supported (0 = no, 1 = yes)

`work_out[6]` Performance rate, in 16 x 16 raster-ops per second

`work_out[7]` Contour fill capability (0 = no, 1 = yes)

`work_out[8]` Text rotation capability :-
- 0 – none
- 1 – 90 degree angles only
- 2 – arbitrary angles

`work_out[9]` Number of available writing modes

`work_out[10]` Input modes available :-
- 0 – none
- 1 – request
- 2 – sample and request

`work_out[11]`	Text alignment capability (0 = no, 1 = yes)
`work_out[12]`	Inking capability (0 = no, 1 = yes)
`work_out[13]`	Rubber banding capability :- 0 – none 1 – lines only 2 – lines and rectangles
`work_out[14]`	Maximum vertices for polyline, polymarker or filled polygon (–1 = no limit)
`work_out[15]`	Maximum size of `VDI_intin` array (–1 = no limit)
`work_out[16]`	Number of keys available on the mouse
`work_out[17]`	Styles work on wide lines (0 = no, 1 = yes)
`work_out[18]`	Writing modes available for wide lines (0 = no, 1 = yes)
`work_out[19]`	Clipping currently enabled (0 = no, 1 = yes) (In GEM versions 2.0 and later)
`work_out[20]` to `work_out[44]`	Reserved (contain zeros)
`work_out[45]`	X coordinate of upper left corner of clipping rectangle in x axis units. (In GEM versions 2.0 and later)
`work_out[46]`	Y coordinate of upper left corner of clipping rectangle in y axis units. (In GEM versions 2.0 and later)
`work_out[47]`	X coordinate of lower right corner of clipping rectangle in x axis units. (In GEM versions 2.0 and later)
`work_out[48]`	Y coordinate of lower right corner of clipping rectangle in y axis units. (In GEM versions 2.0 and later)
`work_out[49]` to `work_out[56]`	Reserved (contain zeros)

8.1.4 Example

```
WORD screen;
WORD work_out[57];
.
.
  vq_extnd(screen, 1, work_out);
  if (work_out[13] == 2 )
  { /* Can do rubberbands, so use them */
    ...
  }
```

8.2 Inquire Color Representation `vq_color`

This can be used to obtain the intensities of red, green and blue with which a given color index is displayed.

8.2.1 Definition

The Prospero C definition of Inquire Color Representation is :

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

8.2.2 Purpose

This function may be used to determine the red, green and blue intensities associated with the given color index. Two sets of intensities are available, those requested by a call to `vs_color` (section 5.2), or those actually realized, which will normally be the closest available values to those requested.

8.2.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
`index`	*WORD*	**Color index** The index of the color about which the application is inquiring.
`setflag`	*WORD*	**Realized value flag** If the value of this parameter is zero, the information returned is the requested intensities, as passed to the last call of `vs_color` (see section 5.2). If this parameter is 1, the values returned will be the intensities of the color as it is actually displayed on the device.
`rgb`	*WORD[3]*	**Red green blue intensities** This is an array of 3 *WORD*s, whose elements return the intensities in tenths of a percent (0 to 1000) as follows :- `rgb[0]` – Red intensity `rgb[1]` – Green intensity `rgb[2]` – Blue intensity

8.2.4 Example

```
WORD screen;
WORD colors[3];
.
.
  /* Get displayed intensities of color index 1 */
  vq_color(screen, 1, 1, colors);
```

8.3 Inquire Line Attributes `vql_attributes`

This can be used to obtain the current settings of the attributes affecting the way that lines are output.

8.3.1 Definition

The Prospero C definition of Inquire Line Attributes is :

```
void vql_attributes(WORD handle, WORD attrib[6]);
```

8.3.2 Purpose

This function may be used to obtain the current settings of the line type, line color, writing mode, line width, beginning line end style and ending line end style. See the relevant functions in section 5 for how the drawing of lines is affected by the settings of the various attributes. The values are returned in an array[6] of *WORD*, defined as follows :-

`attrib[0]`	line type	see `vsl_type` in section 5.3
`attrib[1]`	line color	see `vsl_color` in section 5.6
`attrib[2]`	writing mode	see `vswr_mode` in section 5.1
`attrib[3]`	line width	see `vsl_width` in section 5.5
`attrib[4]`	beginning style	
`attrib[5]`	ending style	see `vsl_ends` in section 5.7

Note that the original C language bindings supplied with the GEM Programmer's Toolkit do not return the beginning and ending line end styles.

8.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
attrib	*WORD[6]*	**Line attributes** This is an array as described above, used to return the current settings of the line attributes. The values returned are in the same form, and with the same meanings, as those passed to the corresponding set attributes routine described in section 5. The original C language bindings supplied with the GEM Programmer's toolkit do not return the beginning and ending line styles.

8.3.4 Example

```
WORD screen;
WORD attributes[6];
.
.
  /* Get line attributes */
  vql_attributes(screen, attributes);

  /* Select color 3 if necessary */
  if (attributes[1] > 3 )
    attributes[1] = vsl_color(screen, 3);
```

8.4 Inquire Marker Attributes `vqm_attributes`

This can be used to obtain the current settings of the attributes affecting the way that polymarkers are output.

8.4.1 Definition

The Prospero C definition of Inquire Marker Attributes is :

```
void vqm_attributes(WORD handle, WORD attrib[5]);
```

8.4.2 Purpose

This function may be used to obtain the current settings of the marker type, marker color, writing mode, marker height and marker width. See the relevant functions in section 5 for how the drawing of polymarkers is affected by the settings of the various attributes. The attributes are returned in an array[5] of *WORD* as follows :-

`attrib[0]`	marker type	see `vsm_type` in section 5.8
`attrib[1]`	marker color	see `vsm_color` in section 5.10
`attrib[2]`	writing mode	see `vswr_mode` in section 5.1
`attrib[3]`	height	
`attrib[4]`	width	see `vsm_height` in section 5.9

Note that the original C language bindings supplied with the GEM Programmer's Toolkit do not return the marker width.

8.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
attrib	*WORD[5]*	**Marker attributes** This is an array as described above, used to return the state of the various attributes affecting polymarker output. The values returned are in the same form, and with the same meanings, as those passed to the corresponding set attributes routine in section 5. The polymarker width cannot be explicitly set, but is determined by the height selected using the `vsm_height` function described in section 5.9. The original C language bindings supplied with the GEM Programmer's Toolkit do not return the marker width.

8.4.4 Example

```
WORD screen;
WORD attributes[5];
.
.
  /* Get marker attributes */
  vqm_attributes(screen, attributes);

  if (attributes[4] < 5)      /* Not big enough */
     attributes[3] = vsm_height(screen, 10);
```

8.5 Inquire Fill Attributes `vqf_attributes`

This can be used to obtain the current settings of the attributes affecting the way that filled areas are output.

8.5.1 Definition

The Prospero C definition of Inquire Fill Attributes is :

```
void vqf_attributes(WORD handle, WORD attrib[5]);
```

8.5.2 Purpose

This function may be used to obtain the current settings of the fill interior mode, fill interior color, fill interior pattern style, and fill perimeter visibility flag. See the relevant functions in section 5 for how the drawing of filled areas is affected by the settings of the various attributes. The attributes are returned in an array[5] of *WORD* as follows:-

`attrib[0]`	fill interior	see `vsf_interior` in section 5.17
`attrib[1]`	fill color	see `vsf_color` in section 5.19
`attrib[2]`	fill style	see `vsf_style` in section 5.18
`attrib[3]`	writing mode	see `vswr_mode` in section 5.1
`attrib[4]`	perimeter flag	see `vsf_perimeter` in section 5.20

Note that the original C language bindings supplied with the GEM Programmer's Toolkit do not return the fill perimeter visibility flag.

8.5.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
attrib	*WORD[5]*	**Fill attributes** This is an array as described above, used to return the state of the attributes affecting the output of filled areas. The values returned are in the same form, and with the same meanings, as those passed to the corresponding set attributes routine in section 5. The original C language bindings supplied with the GEM Programmer's toolkit do not return the perimeter visibility flag.

8.5.4 Example

```
WORD screen;
WORD attributes[5];
.
.
  /* Get fill attributes */
  vqf_attributes(screen, attributes);
  if (attributes[2] == 4)
    /* Not a nice style - change it*/
    attributes[2] = vsf_style(screen, 3);
```

8.6 Inquire Text Attributes `vqt_attributes`

This can be used to obtain the current settings of the attributes affecting the way that text is output.

8.6.1 Definition

The Prospero C definition of Inquire Text Attributes is :

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

8.6.2 Purpose

This function may be used to obtain the current settings of the text font, text color, text baseline angle, text alignment, and character and character cell sizes. See the relevant functions in section 5 for how the output of text is affected by the settings of the various attributes. The values are returned in an array[10] of *WORD*, defined as follows :-

`attrib[0]`	text face	see `vst_font` in section 5.13
`attrib[1]`	text color	see `vst_color` in section 5.14
`attrib[2]`	baseline	see `vst_rotation` in section 5.12
`attrib[3]`	horiz align	
`attrib[4]`	vert align	see `vst_alignment` in section 5.16
`attrib[5]`	writing mode	see `vswr_mode` in section 5.1
`attrib[6]`	char width	
`attrib[7]`	char height	
`attrib[8]`	cell width	
`attrib[9]`	cell height	see `vst_height` in section 5.11

8.6.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
attrib	*WORD[10]*	**Text attributes** This is an array as described above, used to return the current state of all attributes affecting text output. The values returned are in the same form, and with the same meanings, as those passed to the corresponding set attributes routine in section 5. Elements `attrib[6]` and `attrib[7]` return the current character width and height, while `attrib[8]` and `attrib[9]` return the width and height of a character cell, all in the current coordinate system (see `vst_point` or `vst_height` in section 5.11).

8.6.4 Example

```
WORD screen;
WORD attributes[10];
.
.
  /* Get text attributes */
  vqt_attributes(screen, attributes);

  if (attributes[2] != 0)
    /* stop writing at an angle */
    attributes[2] = vst_rotation(screen, 0);
```

8.7 Inquire Text Extent `vqt_extent`

This can be used to discover the area which would be occupied by a given text string, if output using the current text attributes.

8.7.1 Definition

The Prospero C definition of Inquire Text Extent is :

```
void vqt_extent(WORD handle, const char *astring,
                WORD extent[8]);
```

8.7.2 Purpose

This function may be used to obtain the coordinates of the rectangle which would completely contain the given text if output using the current text attributes. As this takes account of the text baseline rotation (see `vst_rotation` in section 5.12), the rectangle may not be parallel to the x and y axes, and therefore the coordinates of all four vertices are required to define it. The rectangle returned always touches the x and y axes at the lower left and upper left corners respectively.

8.7.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
astring	*const char **	**String** This is a string containing the text whose extent is to be returned.
extent	*WORD[8]*	**Extent bounding rectangle** This is an array containing the coordinates of the four corners which define the text extent rectangl. Each point is described by two consecutive elements of the array; the points are in the order:- Lower left, Lower right, Upper right, Upper left. Note that the values of lower left y co-ordinate and upper left x co-ordinate will always be zero.

8.7.4 Example

```
WORD screen;
WORD extent[8];
WORD string_length;
.
.
  /* Get text extent */
  vqt_extent(screen, "Hello There", extent);
  string_length = extent[4];
  /* Valid only if rotation zero! */
```

8.8 Inquire Character Cell Width `vqt_width`

This can be used to discover the character cell width of a specified character in the current text font.

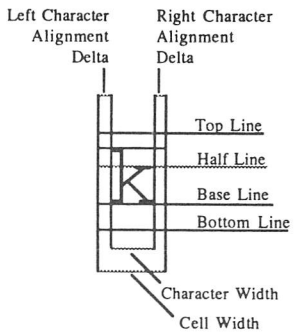
8.8.1 Definition

The Prospero C definition of Inquire Character Cell Width is :

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

8.8.2 Purpose

This function may be used to obtain the width of the specified character in the current text font. The character cell width is the distance from the left hand edge of the character to the left hand edge of the following character in a string. This function does not take account of text special effects or rotation. The function also returns the distance from the left hand edge of the character cell to the left hand edge of the character, and the distance from the right hand edge of the character to the right hand edge of the character cell – these are known as the left and right alignment deltas. All distances are measured in the current coordinate system.



Character Alignment Deltas

8.8.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
character	*WORD*	**Character** The character whose width is to be returned.
cell_width	*WORD* *	**Character cell width** Points to an object which receives the width of the cell enclosing the specified character – the distance between the left hand edge of the character and the left hand edge of the next character in a string.
left_delta right_delta	*WORD* * *WORD* *	**Left character alignment delta** **Right character alignment delta** These parameters points to objects which receive the distances between the left and right hand edges of the character cell and the left and right hand edges of the character.

8.8.4 Function Result

The function returns the ASCII code of the character in question, or –1 if it was not a valid character in the font.

8.8.5 Example

```
WORD screen;
WORD cell, left, right;
.
  vqt_width(screen, 'H', &cell, &left, &right);
```

8.9 Inquire Font Name and Index `vqt_name`

This can be used to discover the name and index of a particular text font.

8.9.1 Definition

The Prospero C definition of Inquire Font Name and Index is :

```
WORD vqt_name(WORD handle, WORD element,
              char name[33]);
```

8.9.2 Purpose

This function may be used to obtain the name and index of a particular text font, which are also referred to as faces in GEM VDI. The font is referred to by its element number – this ranges from 1 to the number of fonts available. However font element 1 is reserved for the system font, which may not always be available on all drivers – this can be determined from the value of `work_out[10]` when a workstation is opened (see section 3.1). The function returns the name of the font, and the font index, which is the value which must be passed to the `vst_font` function described in section 5.13. Font indices are constant for a given font, so that font index 4 is always Swiss 721 Thin Italic. However the element number of this font depends upon what other fonts are listed in ASSIGN.SYS and in what order. This function may be used to determine which fonts are available by calling it for every element number available.

8.9.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
element	*WORD*	**Font element number** This is the element number of the font whose name and index is to be returned.
name	*char [33]*	**Font name** This returns the name and description of the font, as a string of up to 32 characters. The first 16 characters of the string contain the name of the font, padded with blanks. The characters from 17 to the null terminator of the string describe the style, such as light, italic, bold, and so on.

8.9.4 Function Result

The function returns the font index of the font whose element number was specified.

8.9.5 Example

```
WORD screen;
WORD index[20];
char [20][33] names;
int  total_fonts, i;
.
  for ( i = 0; i < total_fonts; i++)
    index[i] = vqt_name(screen, i, names[i]);
```

8.10 Inquire Cell Array `vq_cellarray`

This can be used to return the cell array definition of an area of the display.

8.10.1 Definition

The Prospero C definition of Inquire Cell Array is :

```
void vq_cellarray(WORD handle, WORD xyarray[4],
                  WORD row_length, WORD  num_rows,
                  WORD *el_used, WORD *rows_used,
                  WORD *status, WORD colarray[]);
```

8.10.2 Purpose

This function may be used to return the cell array definition of the specified area of the display. This returns an array, each element of which corresponds to a pixel, and contains the color index of that pixel. The cell array can then be written to another portion of the display using the `v_cellarray` function described in section 4.5.

8.10.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device whose cell array is to be returned.
`xyarray`	*WORD[4]*	**Rectangle** This defines the area of the display whose cell array definition is to be returned, in the standard VDI rectangle form giving the coordinates of two diagonally opposite corners.
`row_length`	*WORD*	**Row length** This defines the length in words of each row in the color index array. Rows of color indices are stored sequentially in the array.

`num_rows` *WORD* **Number of rows**

This defines the number of rows of length `row_length` the color index array contains. The number of elements in the color index array must be at least `row_length * num_rows`.

`el_used` *WORD ** **Elements used per row**

This points to an object which receives the number of pixel values actually placed in each row of the color index array. This will be the width in pixels of the display area defined by `xyarray`, or the value of `row_length`, whichever is smaller. If the number used is less than the length of each row in the color index array, unused elements in each row will be undefined.

`rows_used` *WORD ** **Number of rows used**

This points to an object which receives the number of rows actually placed in the color index array. This will be the height in pixels of the display area defined by `xyarray`, or the value of `num_rows`, whichever is smaller. If the number used is less than the number of rows in the color index array, unused rows will be undefined.

`status` *WORD ** **Error status**

This points to an object which receives non-zero if the color index of some pixel(s) could not be determined (e.g. the pixel does not exist), otherwise zero.

`colarray` *WORD[]* **Color index array**

This is used to pass the array which is to contain the color indices.

8.10.4 Example

```
WORD screen;
WORD rows_used, els_used;
WORD rect[] = { 100, 100, 120, 120 };
WORD problems;
WORD color_array[21][21];
.
.
  vq_cellarray(screen, rect, 21, 21, &els_used,
               &rows_used, &problems, color_array);
  if (!problems)
    {                /* Use color index array */
    ...
    }
```

8.11 Inquire Input Mode `vqin_mode`

This can be used to obtain the input mode (request or sample) currently in use for the specified logical input device.

8.11.1 Definition

The Prospero C definition of Inquire Input Mode is :

```
void vqin_mode(WORD handle, WORD dev_type,
               WORD *mode);
```

8.11.2 Purpose

This function may be used to obtain the current input mode (request or sample) for any of the logical input devices (locator, valuator, choice and string input devices). See section 7 for a description of how the input mode affects the behavior of each input device.

8.11.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device about which the program is inquiring.
`dev_type`	*WORD*	**Logical device type** This is a two-byte integer indicating which of the four logical input devices the application is inquiring about :- 1 – Locator device 2 – Valuator device 3 – Choice device 4 – String device

`mode` *WORD ** **Input mode**

This parameter points to an object which receives the device's input mode :-

1 – request mode
2 – sample mode

8.11.4 Example

```
WORD screen;
WORD mode;
.
.
  /* Get locator input mode */
  vqin_mode(screen, 1, &mode);
```

8.12 Inquire Font Info `vqt_font_info`

This function can be used to obtain size information about the current text font.

8.12.1 Definition

The Prospero C definition of Inquire Font Info is :

```
void vqt_font_info(WORD handle,
                   WORD *min_ADE, *max_ADE,
                   WORD distances[5], WORD *maxwidth,
                   WORD effects[3]);
```

8.12.2 Purpose

This function may be used to obtain information about the size of characters in the currently selected text font, taking into account the current text effects and text size. Height information about the current font is returned in an array[5] of *WORD*, defined as follows :-

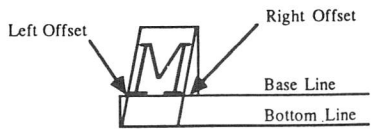
`distance[0]`	depth of bottom line
`distance[1]`	depth of descenders
`distance[2]`	height of half line
`distance[3]`	height of ascent line
`distance[4]`	height of top line

All the above values are in y-axis units measured from the character baseline. See section 5.16 for further information about these values.

The function also returns information about the effect that the current text effects (see section 5.15) have on the characters, in an array[3] of *WORD*, defined as follows :-

`effects[0]`	delta x
`effects[1]`	left offset
`effects[2]`	right offset

The delta x is the increase in character cell width caused by the text effects currently in force. The left offset and the right offset apply to skewed special effects, as shown in the diagram.



These values are in x-axis units.

8.12.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
min_ADE max_ADE	*WORD ** *WORD **	**First character in font** **Last character in font** These point to objects which receive the first and last characters in the font.
distances	*WORD[5]*	**Font distance information** This is an array as described above, returning information about the heights of characters in the font.
maxwidth	*WORD **	**Maximum character cell width** This points to an object which receives the maximum character cell width in the font, in x-axis units, excluding the effects of any special effects such as italics, bolding etc.
effects	*WORD[3]*	**Font effects information** This is an array as described above, returning information about the changes to character widths due to the current text special effects.

8.12.4 Example

```
WORD screen;
WORD min, max;
WORD biggest;
WORD my_distances[5];
WORD my_effects[3];
.
.
  vqt_font_info(screen, &min, &max, my_distances,
                &biggest, my_effects);
  /* Calculate some things from the font information */
```

8.13 Inquire Justified Graphic Text `vqt_justified`

This can be used to obtain the position of each character of a text string output using the `v_justified` function described in section 4.14. This function is not available in GEM version 1.1.

8.13.1 Definition

The Prospero C definition of Inquire Justified Graphic Text is :

```
void vqt_justified(WORD handle, WORD x, WORD y,
                   const char *astring, WORD len,
                   WORD word_space, WORD char_space,
                   WORD offsets[]);
```

8.13.2 Purpose

This function may be used to obtain the x and y coordinates of each character in a text string output using the `v_justified` function (see section 4.14) and the current text attributes.

8.13.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device about which the application is inquiring.
`x` `y`	*WORD* *WORD*	**X coordinate** **Y coordinate** These are used to define the coordinates of the alignment point of the whole string, as passed to `v_justified`.
`astring`	*const char **	**Output string** This contains the string whose character offsets are to be returned, as passed to `v_justified`.

`len` *WORD* **String justification length**

The length to which the string is justified, as passed to `v_justified`.

`word_space` *WORD* **Word space modify flag**
`char_space` *WORD* **Character space modify flag**

These indicate whether the inter-word spacing, the inter-character spacing or both are modified (1 means modify; 0 means don't modify) to justify the text, as passed to `v_justified`.

`offsets` *WORD[]* **Character offsets information**

This parameter specifies the array in which the coordinates of each character will be returned.

8.13.4 Example

```
WORD screen;
WORD char_positions[160];
.
.
  /* Output some text */
  v_justified(screen, 100, 100, "A bit of text",
                                        100, 1, 1);
  /* Find where each character went */
  vqt_justified(screen, 100, 100, "A bit of text",
                        100, 1, 1, char_positions);
```

9 VDI ESCAPES

GEM VDI provides a number of functions which are specific to a particular device or class of device. These are known as escapes, and are all called using the same GEM VDI op-code, with the required function indicated by a sub-opcode (all this is handled internally by the bindings, and need not concern the programmer). Other GEM VDI device drivers may provide additional device dependent escapes, and not all devices will support the escapes described in this section. The escapes described here can be classed according to the devices they are relevant to as follows.

Cursor Addressable Screen Escapes

Section	Function description	Binding name
9.1	Inquire Alpha Character Cells	`vq_chcells`
9.2	Exit Alpha Mode	`v_exit_cur`
9.3	Enter Alpha Mode	`v_enter_cur`

9.4	Move Alpha Cursor	`v_curup`
		`v_curdown`
		`v_curright`
		`v_curleft`

9.5	Home Alpha Cursor	`v_curhome`
9.6	Erase to End of Alpha Screen	`v_eeos`
9.7	Erase to End of Alpha Line	`v_eeol`
9.8	Set Alpha Cursor Address	`vs_curaddress`
9.9	Output Alpha Text	`v_curtext`
9.10	Select Alpha Text Style	`v_rvon` `v_rvoff`
9.11	Inquire Alpha Cursor Address	`vq_curaddress`

General Screen Escapes

Section	Function description	Binding name
9.12	Inquire Tablet Status	`vq_tabstatus`
9.13	Hardcopy	`v_hardcopy`
9.14	Place and Remove Graphic Cursor	`v_dspcur` `v_rmcur`

Printer Escapes

Section	Function description	Binding name
9.15	Form Advance	`v_form_adv`
9.16	Output Window to Printer	`v_output_window`
9.17	Clear Printer Display List	`v_clear_disp_list`
9.18	Output Bit Image File	`v_bit_image_1` `v_bit_image_2`
9.19	Inquire Printer Scan Heights	`vq_scan`
9.20	Output Printer Alpha Text	`v_alpha_text`

Device Specific Escapes

Section	Function description	Binding name
9.21	Select Palette	`vs_palette`
9.22	Generate Tone	`v_sound`
9.23	Set / Clear Muting Flag	`vs_mute`

Tablet Escapes

Section	Function description	Binding name
9.24	Set Tablet Resolution	`vt_resolution` `vt_axis`
9.25	Set Tablet Origin	`vt_origin`
9.26	Inquire Tablet Dimensions	`vq_tdimensions`
9.27	Set Tablet Alignment	`vt_alignment`

GEM Version 2.0 Film Recorder Escapes

Section	Function description	Binding name
9.28	Select Camera Film Type	`vsp_film`
9.29	Inquire Camera Film Name	`vqp_filmname`
9.30	Disable or Enable Film Exposure	`vsc_expose`

GEM Version 1.1 Film Recorder Escapes

Section	Function description	Binding name
9.31	Inquire Film Types	`vqp_films`
9.32	Inquire and Set Palette Driver State	`vqp_state` `vsp_state`
9.33	Save Palette Driver State	`vsp_save`
9.34	Suppress Palette Messages	`vsp_message`
9.35	Palette Error Inquire	`vqp_error`

Metafile Escapes

Section	Function description	Binding name
9.36	Update Metafile Extents	`v_meta_extents`
9.37	Write Metafile Item	`v_write_meta`
9.38	Change GEM VDI Filename	`vm_filename`

9.1 Inquire Alpha Character Cells `vq_chcells`

As well as the graphics (bit mapped) screen supported by the GEM output functions listed in section 4, GEM VDI may be used to send text output to a character based screen (or a screen in character mode). This function can be used to obtain the dimensions in rows and columns of the screen – the following sections detail the operations GEM supports on such a screen.

9.1.1 Definition

The Prospero C definition of Inquire Alpha Character Cells is :

```
void vq_chcells(WORD handle,
                WORD *rows, WORD *columns);
```

9.1.2 Purpose

This function is used to discover the width and height, in characters, of the alphanumeric cursor addressable screen. If cursor addressing is not possible, the width and height returned are –1.

9.1.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the screen device.
rows	*WORD **	**Number of rows in screen** Points to an object which returns the number of rows that can be addressed by the cursor. A value of –1 indicates cursor addressing is not possible.
columns	*WORD **	**Number of columns in screen** Points to an object which returns the number of columns that can be addressed by the cursor. A value of –1 indicates cursor addressing is not possible.

9.1.4 Example

```
WORD screen;
WORD rows, columns;
.
.
  vq_chcells(screen, &rows, &columns);
  if ((rows != -1) && (columns != -1))
    { /* Can do cursor addressing */
      ...
    }
```

9.2 Exit Alpha Mode `v_exit_cur`

This can be used to exit cursor addressable alphanumeric mode on a screen device, and return to graphics mode.

9.2.1 Definition

The Prospero C definition of Exit Alpha Mode is :

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

9.2.2 Purpose

This function is used to change properly from alphanumeric, cursor addressable text mode to graphics mode, if they are different. The graphics screen will be cleared. When a screen workstation is opened, it will be placed in graphics mode.

9.2.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the screen device.

9.2.4 Example

```
WORD screen;
.
.
  v_exit_cur(screen);
  /* Do some graphics output */
```

9.3 Enter Alpha Mode `v_enter_cur`

This function can be used to enter cursor addressable alphanumeric mode, and leave graphics mode.

9.3.1 Definition

The Prospero C definition of Enter Alpha Mode is :

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

9.3.2 Purpose

This function is used to change properly from graphics mode to alphanumeric, cursor addressable text mode, if they are different. The alpha screen will be cleared, and the alpha cursor placed at the home position (top left hand corner).

9.3.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the screen device.

9.3.4 Example

```
WORD screen;
.
.
  v_enter_cur(screen);
  /* Do some alpha type output */
```

9.4 Move Alpha Cursor

v_curup
v_curdown
v_curright
v_curleft

These can be used to move the cursor in alpha mode.

9.4.1 Definition

The Prospero C definitions of the Move Alpha Cursor functions are :

```
void v_curup(WORD handle);
void v_curdown(WORD handle);
void v_curright(WORD handle);
void v_curleft(WORD handle);
```

9.4.2 Purpose

These functions are used to move the alpha cursor one character position up, down, left or right respectively. If the cursor is already at the edge of the screen, nothing happens.

9.4.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the screen device.

9.4.4 Example

```
WORD screen;
int  i;
.
.
  v_enter_cur(screen); /* also homes cursor */
  for (i = 1; i <= 9; i++)
    v_curdown(screen);
  for (i = 1; i <= 9; i++)
    v_curright(screen);/* slow way to move to (10,10)*/
```

9.5 Home Alpha Cursor `v_curhome`

This can be used to move the cursor to the home position in alpha mode.

9.5.1 Definition

The Prospero C definition of Home Alpha Cursor is :

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

9.5.2 Purpose

This function is used to move the alpha cursor to the home position, at the top left hand corner of the screen. Note that the cursor will be placed in this position when alpha mode is entered by the `v_enter_cur` function (section 9.3).

9.5.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the screen device.

9.5.4 Example

```
WORD screen;
.
.
  v_curhome(screen);        /* home cursor */
```

9.6 Erase to End of Alpha Screen `v_eeos`

This can be used to erase the alpha screen from the current cursor position onwards.

9.6.1 Definition

The Prospero C definition of Erase to End of Alpha Screen is :

```
void v_eeos(WORD handle);
```

9.6.2 Purpose

This function is used to erase the alpha screen from the current cursor position onwards. The cursor position is not changed. To erase the entire screen, this could be preceded by a call to `v_curhome` (section 9.5).

9.6.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the screen device.

9.6.4 Example

```
WORD screen;
.
.
  v_curhome(screen);      /* home cursor */
  v_eeos(screen);         /*  and erase entire display */
```

9.7 Erase to End of Alpha Line `v_eeol`

This can be used to erase the line on the alpha screen containing the cursor, from the current cursor position onwards.

9.7.1 Definition

The Prospero C definition of Erase to End of Alpha Line is :

```
void v_eeol(WORD handle);
```

9.7.2 Purpose

This function is used to erase the line containing the cursor on the alpha screen from the current cursor position onwards. This might be used before (or after) outputting text to erase what was previously displayed in that position. The cursor position is not changed.

9.7.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the screen device.

9.7.4 Example

```
WORD screen;
.
.
  v_eeol(screen);          /* erase to end of line */
```

9.8 Set Alpha Cursor Address `vs_curaddress`

This can be used to set the position of the alpha cursor to a particular row and column.

9.8.1 Definition

The Prospero C definition of Set Alpha Cursor Address is :

```
void vs_curaddress(WORD handle, WORD row, WORD col);
```

9.8.2 Purpose

This function is used to set the row and column of the alpha cursor. If the values are beyond the confines of the screen, the nearest value within the screen area will be used.

9.8.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle**
		The handle of the screen device.
row	*WORD*	**Cursor row**
		The row at which the cursor is to be placed.
col	*WORD*	**Cursor column**
		The column at which the cursor is to be placed.

9.8.4 Example

```
WORD screen;
.
.
  vs_curaddress(screen, 10, 10);/* cursor to (10,10) */
```

9.9 Output Alpha Text `v_curtext`

This function can be used to output text to the alpha screen.

9.9.1 Definition

The Prospero C definition of Output Alpha Text is :

```
void v_curtext(WORD handle, const char *astring);
```

9.9.2 Purpose

This function is used to output a text string to the alpha screen at the current cursor position. The current alpha text attribute will be used – see the functions `v_rvon` and `v_rvoff` in section 9.10. Alpha mode must have been entered before text can be output in this manner – see the function `v_enter_cur` in section 9.3. The cursor position will be moved as the text is output.

9.9.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle**
		The handle of the screen device.
`astring`	*const char **	**Output text**
		The text to be output.

9.9.4 Example

```
WORD screen;
.
.
  v_curhome(screen);
  v_curtext(screen, "Top left hand corner");
  v_eeol(screen);       /* Clear rest of top line */
```

9.10 Select Alpha Text Style

v_rvon
v_rvoff

These can be used to select either normal or reverse video text output for the alpha screen.

9.10.1 Definition

The Prospero C definitions of the Select Alpha Text Style functions are :

```
void v_rvon(WORD handle);
void v_rvoff(WORD handle);
```

9.10.2 Purpose

These functions are used to select the style for subsequent alpha text output. The function `v_rvon` causes subsequent alpha text to be output in reverse video, while `v_rvoff` turns off reverse video mode and causes subsequent text output to be in normal video.

9.10.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the screen device.

9.10.4 Example

```
WORD screen;
.
.
  v_rvon(screen);          /* Select reverse video */
  v_curtext(screen, "Reverse Video ");
  v_rvoff(screen);         /* Select normal video */
  v_curtext(screen, "Normal Video ");
```

9.11 Inquire Alpha Cursor Address `vq_curaddress`

This function can be used to discover the position of the alpha cursor.

9.11.1 Definition

The Prospero C definition of Inquire Alpha Cursor Address is :

```
void vq_curaddress(WORD handle,
                   WORD *row, WORD *column);
```

9.11.2 Purpose

This function is used to discover the row and column of the alpha cursor.

9.11.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the screen device.
`row`	*WORD* *	**Cursor row** Points to an object which returns the row at which the cursor is positioned. Row 1 is at the top of the screen.
`column`	*WORD* *	**Cursor column** Points to an object which returns the column at which the cursor is positioned. Column 1 is at the left hand edge of the screen.

9.11.4 Example

```
WORD screen;
WORD row, col;
.
.
  /* get cursor position */
  vq_curaddress(screen, &row, &col);
```

9.12 Inquire Tablet Status `vq_tabstatus`

GEM VDI locator input (see section 7.2) becomes very tedious for the user of a device which does not have a suitable pointer device attached to it, as the cursor must be moved around the screen using the cursor keys. This function can be used to discover whether a tablet, mouse etc. is attached to the specified device.

9.12.1 Definition

The Prospero C definition of Inquire Tablet Status is :

```
WORD vq_tabstatus(WORD handle);
```

9.12.2 Purpose

This function is used to discover whether a graphics tablet, mouse, joystick or similar device is available on the specified device.

9.12.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device.

9.12.4 Function Result

The function returns 1 if a tablet, mouse, or similar device is available, otherwise 0.

9.12.5 Example

```
WORD screen;
.
.
  if (vq_tabstatus(screen))
    { /* Tablet or mouse available */
      ...
    }
```

9.13 Hardcopy v_hardcopy

This can be used to make a hard copy of the specified device to an attached printer or other device.

9.13.1 Definition

The Prospero C definition of Hardcopy is :

```
void v_hardcopy(WORD handle);
```

9.13.2 Purpose

This function is used to copy the contents of the specified device, which should be a screen device, to an attached printer or other hardcopy device. The function is device specific, and may not be supported by all devices.

9.13.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device.

9.13.4 Example

```
WORD screen;
.
.
/* Dump screen image to printer */
v_hardcopy(screen);
```

9.14 Place and Remove Graphic Cursor `v_dspcur` `v_rmcur`

These can be used to place a graphic cursor at a specified location, or to remove the last graphic cursor so placed.

9.14.1 Definition

The Prospero C definitions of Place and Remove Graphic Cursor are :

```
void v_dspcur(WORD handle, WORD x, WORD y);
void v_rmcur(WORD handle);
```

9.14.2 Purpose

The function `v_dspcur` is used to place a graphic cursor at the specified coordinate position The form of the cursor is determined by the function `vsc_form` – see section 7.6. These routines may be used to generate and remove the cursor form for sample mode locator input – see the function `vsm_locator` in section 7.2. These functions are only applicable to devices capable of locator input.

9.14.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle**
		The handle of the device.
`x`	*WORD*	**X coordinate**
`y`	*WORD*	**Y coordinate**
		The x and y coordinates of the point at which the graphic cursor is to be placed.

9.14.4 Example

```
WORD screen
WORD x, y;
.
.
  v_dspcur(screen, x, y);   /* Place a graphic cursor */
  v_rmcur(screen );         /* Remove it */
```

9.15 Form Advance `v_form_adv`

This can be used to advance the printer page.

9.15.1 Definition

The Prospero C definition of Form Advance is :

```
void v_form_adv(WORD handle);
```

9.15.2 Purpose

This function is provided by VDI printer drivers, and causes the printer to advance to a new page, without clearing the printer display list (the printer output which has not yet been sent to the printer). This function may also be used for metafiles, where it will cause a metafile item to be written. It will have no effect on screen devices.

9.15.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the printer or Metafile device.

9.15.4 Example

```
WORD printer;
.
.
  v_form_adv(printer);  /* Advance printer one page */
```

9.16 Output Window to Printer `v_output_window`

This can be used to output a specified portion of the printer display buffer to the printer.

9.16.1 Definition

The Prospero C definition of Output Window to Printer is :

```
void v_output_window(WORD handle, WORD xyarray[4]);
```

9.16.2 Purpose

This function is provided by VDI printer drivers, and causes the specified rectangle of the printer output buffer to be output to the printer. The effect is similar to `v_updwk` (section 3.4), except that a portion of the picture can be specified.

This function will not cause adjacent portions of the picture to be precisely aligned – they will abut within a resolution of one printer head pass height.

This function has no effect on screen devices. Metafile devices will output a metafile item to the buffer.

9.16.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device.
`xyarray`	*WORD[4]*	**Output rectangle** The rectangle defining the area of the picture to be output to the printer. This is in the standard VDI rectangle format, where the elements give the coordinates of two diagonally opposite corners of the rectangle.

9.16.4 Example

```
WORD printer;
WORD rect[] = { 0, 0, 400, 150};
.
.
  /* Output top of picture */
  v_output_window(printer, rect);
```

9.17 Clear Printer Display List `v_clear_disp_list`

This can be used to clear the printer display buffer.

9.17.1 Definition

The Prospero C definition of Clear Printer Display List is :

```
void v_clear_disp_list(WORD handle);
```

9.17.2 Purpose

This function is provided by VDI printer drivers, and causes the printer output buffer to be cleared. The effect is similar to `v_clrwk` (section 3.3), but does not cause the printer to advance to a new page. This function has no effect on screen devices. Metafile devices will output a metafile item to the buffer.

9.17.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the printer device.

9.17.4 Example

```
WORD printer;
.
.
  v_clear_disp_list(printer);/* Clear printer buffer */
```

9.18 Output Bit Image File `v_bit_image_1` `v_bit_image_2`

This function is used to process a bit image file. The function and its binding have been substantially altered between GEM versions 1.1 and 2.0.

9.18.1 Definition

The Prospero C definitions of Output Bit Image File are :

GEM version 1.1 :-

```
void v_bit_image_1(WORD handle, const char *filename,
                   WORD aspect, WORD scaling,
                   WORD num_pts, WORD xyarray[4]);
```

GEM version 2.0 :-

```
void v_bit_image_2(WORD handle, const char *filename,
                   WORD aspectflg, WORD xscale,
                   WORD yscale, WORD h_align,
                   WORD v_align, WORD xyarray[4]);
```

9.18.2 Purpose

This function processes the named bit image file, and draws the processed image on the specified device, which should be a printer or camera device, or a metafile. Various forms of scaling and transformation may be applied to the image in the file before it is output. The function has been substantially altered by Digital Research between GEM versions 1.1 and 2.0, and therefore two versions of the binding are supplied for the two versions of GEM.

9.18.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the bit image is to be output.
filename	*const char **	**Bit image file name** The filename and directory specification of the bit image file. Bit image files have an extension .IMG.
aspect	*WORD*	**Aspect ratio treatment** In GEM version 1.1, the image aspect ratio may be processed in one of three ways, according to the value of this parameter :- 0 – ignore aspect ratio 1 – honor pixel aspect ratio (e.g. circles remain round). 2 – honor page aspect ratio, so full page in bit image space maps to a full page on the output device.
scaling	*WORD*	**Scaling treatment** In GEM version 1.1, the image may be scaled in two manners as follows:- 0 – uniform scaling in x and y axes – the image may not completely occupy the specified output rectangle, but the aspect ratio parameter will be obeyed. 1 – separate scaling in x and y axes – the image will completely fill the specified output rectangle, but the aspect ratio parameter will be ignored. This parameter is only relevant when the value of the parameter num_pts is 2, indicating that both upper left and lower right corners of the output rectangle have been specified.

`num_pts` *WORD* **Number of points specified**

In GEM version 1.1, the rectangle to which the image is to be output may be specified in one of 3 ways :-

0 – use upper left and lower right points from image file header. The points specified in the parameter `xyarray` are not used.

1 – use upper left point specified in `xyarray[0]` and `xyarray[1]` and lower right point specified in image file header. The values of `xyarray[2]` and `xyarray[3]` are not used.

2 – use upper left and lower right points from the parameter `xyarray`. The `aspect` parameter is not relevant, but the `scaling` parameter is.

`aspectflg` *WORD* **Aspect ratio flag**

In GEM version 2.0, either the aspect ratio may be ignored (`aspectflg = 1`) or the pixel aspect ratio may be preserved (`aspectflg = 0`).

`xscale` *WORD* **X axis scaling flag**
`yscale` *WORD* **Y axis scaling flag**

In GEM version 2.0, the scaling of the two axes may be specified separately as either fractional (`scale = 0`), or integer (`scale = 1`). Fractional scaling will cause the image to precisely fill the specified rectangle, while integer scaling may not – the image size can only be scaled in integer multiples. Note that some combinations of the scaling and aspect ratio parameters may cause the scaled image to exceed the size of the scaling rectangle. In this case the bit image will be clipped to the scaling rectangle.

`h_align` *WORD* **Horizontal alignment**
`v_align` *WORD* **Vertical alignment**

In GEM version 2.0, if the scaled image does not precisely fit in the scaling rectangle, the alignment within the rectangle may be specified using these parameters :-

0 – left/top alignment
1 – centre alignment
2 – right/bottom alignment

`xyarray` *WORD[4]* **Scaling rectangle**

The rectangle to which the scaled image is to be output, in the standard GEM VDI format. In GEM version 1.1, the number (0, 1, or 2) of points in this array which are actually used is specified by the parameter `num_pts`.

9.18.4 Example

```
WORD printer;
char *imagefile;
WORD rect[4];
.
.
  /* GEM 1.1 version */
  /* Output image, scaling and rect irrelevant as */
  /* num_pts is zero */
  v_bit_image_1(printer, imagefile, 0, 0, 0, rect);

  /* GEM 2.0 version */
  rect[0] = 0;
  rect[1] = 0;
  rect[2] = 200;
  rect[3] = 400;
  /* Output image, no scaling */
  v_bit_image_2(printer, imagefile, 0, 1, 1,
                0, 0, rect);
```

9.19 Inquire Printer Scan Heights `vq_scan`

This function is used to discover the printer head scan height. It is not provided in GEM version 1.1.

9.19.1 Definition

The Prospero C definition of Inquire Printer Scan Heights is :

```
void vq_scan(WORD handle, WORD *g_height,
             WORD *g_slices, WORD *a_height,
             WORD *a_slices, WORD *factor);
```

9.19.2 Purpose

This function returns information about the height of each printer head scan, in both alpha and graphics mode. This function is not provided in GEM version 1.1.

9.19.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the printer device.
g_height	*WORD **	**Graphics mode scan height** Points to an object which returns the height of a graphics mode printer scan, in scaled pixels.
g_slices	*WORD **	**Graphics mode passes per page** Points to an object which returns the number of printer head passes required to output an entire page, in graphics mode.
a_height	*WORD **	**Alpha mode scan height** Points to an object which returns the height of an alpha mode printer scan, in scaled pixels.

`a_slices` *WORD ** **Alpha mode passes per page**

Points to an object which returns the number of printer head passes required to output an entire page, in alpha mode.

`factor` *WORD ** **Division factor**

Points to an object which returns a value by which the scan heights returned in `g_height` and `a_height` should be divided, so that non-integral scan heights may be returned.

9.19.4 Example

```
WORD printer;
WORD gh, gs, ah, as, factor;
.
  vq_scan(printer, &gh, &gs, &ah, &as, &factor);
  if (gh < 10*factor)
    { /* graphic scan height less than 10 */
      ...
    }
```

9.20 Output Printer Alpha Text `v_alpha_text`

This function is used to output a string of text to the printer. This function is not provided in GEM version 1.1.

9.20.1 Definition

The Prospero C definition of Output Printer Alpha Text is :

```
void v_alpha_text(WORD handle, const char *astring);
```

9.20.2 Purpose

This function outputs the specified text to the printer at the current printer head position. A printer font must have been loaded before this is used – see `vst_load_fonts` in section 3.5. Characters are output as specified, except for the following special codes :-

Form Feed ('\12') has the same effect as calling `v_form_adv` (section 9.15)

DC2 ('\18') followed by a character in the range 0 to 5 modifies the style as follows :-

0 – begin boldface
1 – end boldface
2 – begin italics
3 – end italics
4 – begin underline
5 – end underline

This function is not provided in GEM version 1.1.

9.20.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the printer device.
astring	*const char* *	**Output string** The string of alpha text to be output.

9.20.4 Example

```
WORD printer;
.
.
  v_alpha_text(printer, "This goes to the printer");
```

9.21 Select Palette `vs_palette`

This function is used to select one of the two available palettes on the IBM medium resolution video display.

9.21.1 Definition

The Prospero C definition of Select Palette is :

```
WORD vs_palette(WORD handle, WORD palette);
```

9.21.2 Purpose

This function is specific to the IBM medium resolution screen, and selects one of the two palettes, so that either red, green and brown or cyan, magenta and white are available.

9.21.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the screen device.
palette	*WORD*	**Palette selected** The palette required :- 0 – red, green, brown palette (default) 1 – cyan, magenta, white palette

9.21.4 Function Result

The function returns the palette selected (0 or 1).

9.21.5 Example

```
WORD screen;

  /* Select cyan palette */
  vs_palette(screen, 1);
```

9.22 Generate Tone `v_sound`

This function is used to generate a tone. It is not available in GEM version 1.1.

9.22.1 Definition

The Prospero C definition of Generate Tone is :

```
void v_sound(WORD handle, WORD frequency,
             WORD duration);
```

9.22.2 Purpose

This function generates a tone of the specified frequency for the specified duration. Tone generation may be suppressed by the function `vs_mute` (see section 9.23). This function is not provided in GEM version 1.1.

9.22.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device.
`frequency`	*WORD*	**Tone frequency** The frequency of the required tone, in Hertz.
`duration`	*WORD*	**Tone duration** The duration of the required tone, in timer ticks.

9.22.4 Example

```
WORD screen;
int  i;
.
.
  for (i = 50; i <= 400; i++)
    v_sound(screen, i, 10);     /* Glissando */
```

9.23 Set/Clear Muting Flag `vs_mute`

This function is used enable or disable tone generation, or to discover the current setting of the muting flag. It is not available in GEM version 1.1.

9.23.1 Definition

The Prospero C definition of Set/Clear Muting Flag is :

```
WORD vs_mute(WORD handle, WORD action);
```

9.23.2 Purpose

This function allows an application to modify or discover the state of the muting flag. When the muting flag is set, tone generation by the `v_sound` function (section 9.22) is disabled. This function is not provided in GEM version 1.1.

9.23.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device.
`action`	*WORD*	**Action to perform** This parameter indicates whether the muting flag is to be set, cleared, or returned unmodified as follows :- –1 – Return state of muting flag unmodified 0 – Enable tone generation (muting off) 1 – Disable tone generation (muting on)

9.23.4 Function Result

The function returns the state of the muting flag. A value of zero indicates that muting is off, and tones can be generated. Any other value means muting is on.

9.23.5 Example

```
WORD screen;
.
.
  if (vs_mute(screen, -1) == 0)
    {                    /* Tone generation enabled */
    /* Make some noise */
```

9.24 Set Tablet Resolution `vt_resolution` `vt_axis`

GEM version 2.0 includes a number of functions for use with a device driving a graphics tablet, described here and in the following sections. These functions are used to set the resolution of a graphics tablet. They are not available in GEM version 1.1.

9.24.1 Definition

The Prospero C definitions of Set Tablet Resolution are :

```
void vt_resolution(WORD handle, WORD xres, WORD yres,
                   WORD *xset, WORD *yset);

void vt_axis(WORD handle, WORD xres, WORD yres,
             WORD *xset, WORD *yset);
```

9.24.2 Purpose

These functions allow an application to set the horizontal and vertical resolutions of an attached graphic tablet in either lines per inch or lines per axis. The resolutions are set to the nearest available values, and the actual values set are returned. These functions are not provided in GEM version 1.1.

9.24.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the tablet is attached.
xres yres	*WORD* *WORD*	**X axis resolution requested** **Y axis resolution requested** The requested values of the x and y axis resolutions, in lines per inch (for vt_resolution) or lines per axis (for vt_axis).
xset yset	*WORD ** *WORD **	**X axis resolution selected** **Y axis resolution selected** These parameters point to objects which return the values of the x and y axis resolutions actually selected in lines per inch (for vt_resolution) or lines per axis (for vt_axis).

9.24.4 Example

```
WORD tablet;
WORD xs, ys;
.
.
  /* Request 100 lines per inch resolution */
  vt_resolution(tablet, 100, 100, &xs, &ys);
  if ((xs < 50) || (ys < 50))
    {
    /* Tablet has insufficient resolution */
    ...
```

9.25 Set Tablet Origin `vt_origin`

This function is used to set the origin of a graphics tablet. It is not available in GEM version 1.1.

9.25.1 Definition

The Prospero C definition of Set Tablet Origin is :

```
void vt_origin(WORD handle,
               WORD xorigin, WORD yorigin);
```

9.25.2 Purpose

This function allows an application to set the x and y coordinates of the tablet origin, in lines (tablet units). It is not provided in GEM version 1.1.

9.25.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the tablet is attached.
xorigin	*WORD*	**Origin X coordinate**
yorigin	*WORD*	**Origin Y coordinate** The coordinates of the required origin. These are in tablet units as set by the `vt_resolution` or `vt_axis` rather than the NDC or Raster device coordinates.

9.25.4 Example

```
WORD tablet;
WORD xs, ys;

  /* Select 100 lines per inch */
  vt_resolution(tablet, 100, 100, &xs, &ys);
  vt_origin(tablet, xs, 0); /* Move origin by 1 inch */
```

9.26 Inquire Tablet Dimensions `vq_tdimensions`

This function is used to discover the dimensions of a graphics tablet. It is not available in GEM version 1.1.

9.26.1 Definition

The Prospero C definition of Inquire Tablet Dimensions is :

```
void vq_tdimensions(WORD handle,
                    WORD *xsize, WORD *ysize);
```

9.26.2 Purpose

This function allows an application to discover the x and y dimensions of the tablet in tenths of inches. It is not provided in GEM version 1.1.

9.26.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the device to which the tablet is attached.
xsize	*WORD **	**Tablet X dimension**
ysize	*WORD **	**Tablet Y dimension** These parameters point to objects which return the dimensions of the tablet surface, in tenths of inches.

9.26.4 Example

```
WORD tablet;
WORD xs, ys;

  vq_tdimensions(tablet, &xs, &ys);
                                  /* Get tablet size */
  if ( (xs < 100) || (ys < 100) )
    {
    /* Tablet less than 10x10 inches*/
```

9.27 Set Tablet Alignment `vt_alignment`

This function is used to establish an alignment between the coordinate axes of the tablet, and those of a drawing to be traced. It is not available in GEM version 1.1.

9.27.1 Definition

The Prospero C definition of Set Tablet Alignment is :

```
void vt_alignment(WORD handle, WORD dx, WORD dy);
```

9.27.2 Purpose

This function can be used to rotate the axes used by the tablet, for example to make them correspond precisely to those of a drawing about to be traced. The line along which the axis is to lie is specified by giving the x and y components of its slope in the current tablet coordinate system. These will typically be obtained by asking the user to select on the tablet the two end points of the desired axis. If these points are (x1, y1) and (x2, y2), the values to be passed in `dx` and `dy` to make this line correspond to an axis are (x2–x1) and (y2–y1).

This function is not provided in GEM version 1.1.

9.27.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the device to which the tablet is attached.
`dx` `dy`	*WORD* *WORD*	**X element of slope** **Y element of slope** These parameters give the signed length of the x or y component of the slope (i.e. of the sides of the right angled triangle whose hypotenuse lies along the required axis).

9.27.4 Example

```
WORD tablet;
WORD p1[2], p2[2];
.
.
  /* Get end points of axis from user in p1, p2 */
  vt_alignment(handle, p2[0] - p1[0], p2[1] - p1[1]);
```

9.28 Select Camera Film Type `vsp_film`

GEM versions 1.1 and 2.0 provide a number of escapes for use with a film recorder device, though the escapes provided have completely changed between the two releases of GEM. The film recorder functions provided in GEM version 2.0 are described in sections 9.28 to 9.30, while those for GEM version 1.1 are described in sections 9.31 to 9.35.

This function is used to select the film type and exposure for a camera device under GEM version 2.0.

9.28.1 Definition

The Prospero C definition of Select Camera Film Type is :

```
void vsp_film(WORD handle, WORD index, WORD lightness);
```

9.28.2 Purpose

This function is used in GEM version 2.0 to select the film type and exposure for film recorder devices.

9.28.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the film recorder device.
`index`	*WORD*	**Film index** The index of the film to be used. This should be between one and the number of films available. The function `vqp_filmname` (section 9.29) can be used to find the name of the film associated with each index.

lightness *WORD* **Exposure index**

An integer in the range –3 to 3, giving the aperture to be used in units of a third of an f-stop. A value of –3 results in half the normal exposure, while +3 will result in double the normal exposure.

9.28.4 Example

```
WORD camera;
.
.
  /* Select film index 1, normal exposure */
  vsp_film(camera, 1, 0);
```

9.29 Inquire Camera Film Name `vqp_filmname`

This function is used to return the name of the film associated with the given film index for a camera device. It is not available in GEM version 1.1, which has different camera functions.

9.29.1 Definition

The Prospero C definition of Inquire Camera Film Name is :

```
WORD vqp_filmname(WORD handle, WORD index,
                  char name[]);
```

9.29.2 Purpose

This function is used in GEM version 2.0 to discover the name of a specified film index in a camera device.

9.29.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the film recorder device.
index	*WORD*	**Film index** The index of the film whose name is to be returned.
name	*char[]*	**Film name** An array to receive the name of the specified film. This should have space for at least 25 characters, including the null terminating character.

9.29.4 Function Result

This function returns non-zero if the film index was valid and a valid film name has been returned, otherwise zero.

9.29.5 Example

```
WORD camera;
char name[25];
.
.
  if (vqp_filmname(camera, 1, name))
    /* Film index 1 valid, so use it */
    vsp_film(camera, 1, 0);
```

9.30 Disable or Enable Film Exposure `vsc_expose`

This function is used to disable or enable exposure on camera devices which support previewing of the image to be exposed. It is not available in GEM version 1.1, which has different camera functions.

9.30.1 Definition

The Prospero C definition of Disable Or Enable Film Exposure is :

```
void vsc_expose(WORD handle, WORD state);
```

9.30.2 Purpose

This function is used in GEM version 2.0 to enable or disable exposure of an image so that the image can be previewed before the exposure is made. This function only applies to camera devices which support previewing of images.

9.30.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the film recorder device.
state	*WORD*	**Exposure flag** If this parameter is 1, exposure of images is enabled. This is the default state. If this parameter is 0, exposure is disabled, and previewing of the image may be used.

9.30.4 Example

```
WORD camera;
.
.
  /* Disable exposure for preview */
  vsc_expose(camera, 0);
```

9.31 Inquire Film Types `vqp_films`

The functions described in sections 9.31 to 9.35 are the GEM version 1.1 functions for use with the Polaroid Palette film recorder device. They have been replaced under GEM version 2.0 with the functions described in sections 9.28 to 9.30.

This function is used to return the names of the 5 films which the camera device is currently capable of exposing. This function is only available in GEM version 1.1 – in GEM version 2.0 it has been superseded by the function `vqp_filmname` (section 9.29).

9.31.1 Definition

The Prospero C definition of Inquire Film Types is :

```
void vqp_films(WORD handle,char filmnames[125]);
```

9.31.2 Purpose

This function is used in GEM version 1.1 to return the names of the five films with which the camera is currently loaded.

9.31.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the film recorder device.
`filmnames`	*char [125]*	**Film names** This parameter returns the names of the five films currently loaded in the camera device.

9.31.4 Example

```
WORD camera;
char names[5][25];
char * requiredfilm = "Kodachrome ASA 100        ";
                                        /* e.g. */
int  i;
.
.
  vqp_films(camera, names);      /* Get film names */
  i = 0;
  while (i < 5)
    if (strcmp (names[i], requiredfilm) == 0)
    /* Select that film, and set i to 5 to stop loop */
    else
      i++;
```

9.32 Inquire and Set Palette Driver State `vqp_state` `vsp_state`

These functions are used to return or alter the current state of the Polaroid Palette film recorder. They are only available in GEM version 1.1.

9.32.1 Definition

The Prospero C definitions of Inquire Film Types State are :

```
void vqp_state(WORD handle, WORD *port,
               WORD *film_num, WORD *lightness,
               WORD *interlace, WORD *planes,
               WORD indexes[16]);

void vsp_state(WORD handle, WORD port,
               WORD film_num, WORD lightness,
               WORD interlace, WORD planes,
               WORD indexes[16]);
```

9.32.2 Purpose

These functions are used in GEM version 1.1 to set or return information about the current state of the film recorder. Normally, an application will obtain the current state of the driver using `vqp_state`, modify one or more of the variables in which the state was returned, then pass the same variables to `vsp_state` to alter the palette state. In GEM 2.0 the function `vsp_film` is used to perform some of the function of `vsp_state` – see section 9.28.

9.32.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the film recorder device.
`port`	*WORD* *WORD **	**Port number** This parameter returns or sets the number of the communications port to which the recorder is connected. A value of zero indicates the first communications port, and so on.
`film_num`	*WORD* *WORD **	**Film number** This parameter returns or sets the film to be used, in the range 0 to the number of films available. See the function `vqp_films` for how to discover what films are available.
`lightness`	*WORD* *WORD **	**Aperture control** This parameter returns or sets the lightness to be used for exposing an image, in units of a third of an f-stop. A value of 0 indicates normal exposure, a value of 3 will double the exposure, and –3 will halve it.
`interlace`	*WORD* *WORD **	**Interlace flag** This parameter returns or sets the state of the interlace flag. If interlace is enabled (`interlace` = 1), the picture will require twice as much memory to store.
`planes`	*WORD* *WORD **	**Number of planes** This parameter returns or sets the number of planes to be used – this should be in the range 1 to 4, giving 2, 4, 8, or 16 colors.

indexes	*WORD[16]*	**Color indices**

This parameter returns or sets the color indices for up to 8 colors. Each color is referred to by a pair of characters, a letter in the range 'A' to 'H' and a number in the range '1' to '9', specifying a column and row in the film's color table. The parameter is declared as an array[16] of *WORD*; each of the 8 pairs of words corresponds to a single color. The ASCII codes of the relevant letters should be placed in the *WORD* elements.

9.32.4 Example

```
WORD camera;
WORD port, index, aperture, planes;
WORD interlace;
WORD colors[16];
.
.
  /* Get palette state */
  vqp_state(camera, &port, &index, &aperture,
            &interlace, &planes, colors);
  /* Select a suitable film */
  if (index > 3) index = 1;

  /* Set new palette state */
  vsp_state(camera, port, index, aperture, interlace,
            planes, colors);
```

9.33 Save Palette Driver State `vsp_save`

This function is used to save to disk the current state of the Polaroid Palette film recorder, and thus cause it to be the default state in future. It is only available in GEM version 1.1.

9.33.1 Definition

The Prospero C definition of Save Palette Driver State is :

```
void vsp_save(WORD handle);
```

9.33.2 Purpose

This function is used in GEM version 1.1 to save the current state of the film recorder as the default state. There is no equivalent function in GEM version 2.0.

9.33.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the film recorder device.

9.33.4 Example

```
WORD camera;
.
.
  /* Select a palette state */
  /* Make it the default state */
  vsp_save(camera);
```

9.34 Suppress Palette Messages `vsp_message`

This function is used to suppress the output of messages to the screen by the film recorder device. It is only available in GEM version 1.1.

9.34.1 Definition

The Prospero C definition of Suppress Palette Messages is :

```
void vsp_message(WORD handle);
```

9.34.2 Purpose

This function is used in GEM version 1.1 to suppress the output of messages directly to the screen by the film recorder. If this is used, palette messages may be checked for (and appropriate action taken using the function `vqp_error` (section 9.35). There is no equivalent function in GEM version 2.0.

9.34.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the film recorder device.

9.34.4 Example

```
WORD camera;
.
.
  vsp_message(camera);     /* Suppress message output */
```

9.35 Palette Error Inquire `vqp_error`

This function is used to discover whether the film recorder device has any errors or user prompts pending. It is only available in GEM version 1.1.

9.35.1 Definition

The Prospero C definition of Palette Error Inquire is :

```
WORD vqp_error(WORD handle);
```

9.35.2 Purpose

This function is used in GEM version 1.1 to discover whether any errors have been detected by the film recorder, or whether it has any user prompts pending. It does not clear the error or prompt condition, so that unless `vsp_message` has been used (see section 9.34) the relevant message will still be output to the screen. However it is more usual to use this function when normal error reporting has been disabled by `vsp_message`, so that an application can inform the user of any errors or any action required. There is no equivalent function in GEM version 2.0.

9.35.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the film recorder device.

9.35.4 Function Result

The function returns a `WORD` indicating which error or prompt is being reported as follows :-

0 – no error
1 – open dark slide for print film
2 – no port at location specified
3 – palette not found at specified port
4 – video cable disconnected
5 – operating system does not allow memory allocation
6 – not enough memory to allocate buffer
7 – memory not de-allocated
8 – driver file not found
9 – driver file is incorrect type
10 – prompt user to process print film

9.35.5 Example

```
WORD camera;
WORD error;
.
.
  error = vqp_error(camera);/* check for errors etc. */
  if (error != 0)
    switch (error) {
      ...
```

9.36 Update Metafile Extents `v_meta_extents`

This function may be used to set the extents information in a metafile header.

9.36.1 Definition

The Prospero C definition of Update Metafile Extents is :

```
void v_meta_extents(WORD handle,
                    WORD min_x, WORD min_y,
                    WORD max_x, WORD max_y);
```

9.36.2 Purpose

This function is used to set the extents information in a metafile header. This can then be used by an application loading the metafile as an indication of the minimum rectangle that encloses all the graphics primitives output to the metafile. If this function is not used, all coordinates of the extents record will be zero.

9.36.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
handle	*WORD*	**Device handle** The handle of the metafile device.
min_x	*WORD*	**Minimum X coordinate**
min_y	*WORD*	**Minimum Y coordinate**
max_x	*WORD*	**Maximum X coordinate**
max_y	*WORD*	**Maximum Y coordinate** The coordinates defining the bounding rectangle enclosing all elements in the metafile. Metafile devices have coordinates in the range 0 to 32767 in both NDC and RC modes, although they have different origins.

9.36.4 Example

```
WORD metafile;
.
.
  v_meta_extents(metafile, 0, 0, 1000, 1000);
```

9.37 Write Metafile Item `v_write_meta`

This function may be used write a user defined item to a metafile.

9.37.1 Definition

The Prospero C definition of Write Metafile Item is :

```
void v_write_meta(WORD handle, WORD num_intin,
                  WORD intin[], WORD num_ptsin,
                  WORD ptsin[]);
```

9.37.2 Purpose

This function is used to write the contents of the `intin` and `ptsin` arrays (cf. section 1) to a metafile, with an op-code identifying the item as a user defined item. The sub op-code in `intin[0]` should contain a value 101 or higher identifying the type of user defined item.

9.37.3 Parameters

`Parameter name`	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the metafile device.
`num_intin`	*WORD*	**Size of intin array** The number of elements in the specified `intin` array. There should always be at least one point, as `intin[0]` should hold the metafile sub-opcode.

`intin` *WORD[]* **Intin array**

A pointer to the `intin` array, which contains `num_intin` elements of type *WORD* to be written to the metafile. The first element `intin[0]` should always contain the metafile sub-opcode in the range 101 and above, identifying the type of this user defined metafile item. Other elements contain user defined *WORD* values describing the item.

`num_ptsin` *WORD* **Size of ptsin array**

The number of coordinate pairs in the specified `ptsin` array.

`ptsin` *WORD[]* **Ptsin array**

A pointer to the `ptsin` array, which contains `num_ptsin * 2` elements of type *WORD* to be written to the metafile. No values will be written if the value of `num_ptsin` is zero. The elements contain user defined coordinate pairs describing the item.

9.37.4 Example

```
WORD metafile;
WORD intin[5];
WORD ptsin[4]; /* 2 coordinate pairs */
.
.
  /* Set up intin and ptsin arrays */
  intin[0] = 101;
                  /* Sub opcode of this metafile item */
  v_write_meta(metafile, 5, intin, 2, ptsin);
```

9.38 Change GEM VDI Filename `vm_filename`

This function may be used change the name of a metafile device.

9.38.1 Definition

The Prospero C definition of Change GEM VDI Filename is :

```
void vm_filename(WORD handle, const char *filename);
```

9.38.2 Purpose

This function is used to change the filename and/or pathname of the metafile device. It must be called immediately after opening the metafile using `v_opnwk` (section 3.1), or it will have no effect. Any open metafiles will be closed and deleted. Metafiles always have an extension .GEM. If this function is not used, the metafile name will be GEMFILE.GEM.

9.38.3 Parameters

Parameter name	*Type of parameter*	**Parameter description** Function of parameter
`handle`	*WORD*	**Device handle** The handle of the metafile device.
`filename`	*const char **	**Metafile path and file name** The pathname and filename to be used for the metafile.

9.38.4 Example

```
WORD metafile;
.
.
  vm_filename(metafile, "C:\MYMETA");
```

10 INDEX OF FUNCTIONS

Prospero Software

LANGUAGES FOR MICROCOMPUTER PROFESSIONALS

Prospero C VDI Bindings