The Virtual Device Interface

In this document, I list assembly language and C interfaces to the VDI. The C prototypes are similar to those provided by the GEM Developer's Kit, version 3.0. Most of the C interfaces are declared to return WORDs, but in practice only a few of them return meaningful values. The symbol  indicates that a function behaves differently in different GEM versions.

Many of the low-numbered functions (eg: the "Cell array" functions, and the "locator/valuator/choice" input devices) are not implemented in most GEM drivers; they are relics of GSX and GKS.

This is still a work in progress. Some information is missing or requires additional explanation.

The VDI is used by GEM to communicate with its graphics drivers. It is used for all drawing functions.

The VDI is a superset of the Graphics System eXtension interface. GEM/2 for the PC provides a GSX interface; GEM/3 does not. It is made up of the GDOS, which loads drivers and performs Bezier drawing functions and coordinate transformations, and the drivers. Nearly all VDI functions passed to the drivers.

The GDOS lives in the file GEMVDI.EXE (in ViewMAX, VIEWMAX.EXE). GEM (but not ViewMAX) allows the VDI to use alternative AES modules:

```
GEMVDI /aesfile
```

This can be used to load the VDI without the AES:

```
GEMVDI /C:\COMMAND.COM
```

will load COMMAND.COM and wait until it terminates (ie, EXIT is typed).

The VDI is accessed by using INT 0EFh with CX=0473h and DS:DX=address of parameter block. The value in CX is not checked by the VDI, but should be used to ensure that the AES - or any other program chaining this interrupt - does not attempt to handle the call.

The parameter block format is:

```
        DD      CONTRL      ;Address of control array
        DD      INTIN       ;Address of integer input array
        DD      PTSIN       ;Address of pixel input array
        DD      INTOUT      ;Address of integer output array
        DD      PTSOUT      ;Address of pixel output array
```

The control array is:

```
CONTRL:
        DEFW    function    ;Input:  Function number
        DEFW    #ptsin      ;Input:  Number of points in PTSIN array.
        DEFW    #ptsout     ;Output: Number of points in PTSOUT array.
        DEFW    #intin      ;Input:  Number of integers in INTIN array.
        DEFW    #intout     ;Output: Number of integers in INTOUT array.
XCTRL:
        DEFW    special     ;Input:  For special uses.
        DEFW    handle      ;GEM driver handle
        DD      ptr1        ;First far pointer value
        DD      ptr2        ;Second far pointer value
```

The coordinates used in PTSIN and PTSOUT are signed 16-bit integer values, 0-7FFFh - The *x* value comes first. Normally these are *device units* - the actual pixels on the screen, printer etc. GEM can also be set to scale the coordinates, so that 0-7FFFh is the full height or width of the screen (as in GSX).

In all calls except those where it is explicitly stated otherwise, *handle* is passed to the VDI on entry. If values for *#ptsin* and *#intin* are not supplied, these should be set to 0 on entry.

Function 1 - Open workstation

C Interface:

- WORD v_opnwk(WORD work_in[], WORD *handle, WORD work_out[]);

Entered with:

- *function*=1;
- *#ptsin*=0;
- *#intin*=11;
- INTIN contains initial settings.

The supplied bindings set *#intin* to 103. But the sample programs only supply 11 values.

Returns:

- *#ptsout*= 6;
- *#intout*= 45;
- *handle* = device handle, 0 if none available.
- INTOUT and PTSOUT contain device data (in the C interface, the INTOUT data appear at work_out[0] and the PTSOUT data at work_out[45]).

This loads a device driver and initialises it. Valid device driver numbers are found in the ASSIGN.SYS file (GEM/1 or 2; they are assigned automatically in GEM/3 and later), and follow this pattern:

```
 1- 9: Screen
11-19: Plotter
21-29: Printer
31-39: GEM metafile.
```

The format of the INTIN array is:

```
INTIN:  DEFW        device_number
        DEFW        line_style
        DEFW        line_colour
        DEFW        marker_style
        DEFW        marker_colour
        DEFW        text_style
        DEFW        text_colour
        DEFW        fill_style
        DEFW        fill_index
        DEFW        fill_colour
        DEFW        pixel_mode  ;1 to use NDCs, 2 to use pixels
```

The INTOUT return array gives:

```
INTOUT: DEFW        Screen width, device units
        DEFW        Screen height, device units
        DEFW        0 if device can produce a precisely scaled image,
```

```
                1 if it can't
DEFW            Width of a pixel, in thousandths of a millimetre.
DEFW            Height of a pixel, in thousandths of a millimetre.
DEFW            Number of character sizes, 0 for continuous sizing.
DEFW            Number of line styles.
DEFW            Number of line widths, 0 for continuous scaling.
DEFW            Number of marker styles.
DEFW            Number of marker sizes, 0 for continuous scaling.
DEFW            Number of fonts.
DEFW            Number of patterns.
DEFW            Number of hatch styles.
DEFW            Number of colours displayable at once.
DEFW            Number of General Drawing Primitives
DEFS            20  ;General Drawing Primitive list.
                    ;-1 => End of list
                    ; 1 => Bar
                    ; 2 => Arc
                    ; 3 => Pie slice
                    ; 4 => Circle
                    ; 5 => Ellipse
                    ; 6 => Elliptical arc
                    ; 7 => Elliptical pie slice
                    ; 8 => Rounded rectangle
                    ; 9 => Filled rounded rectangle
                    ;10 => Justified graphic text
DEFS            20        ;General Drawing Primitive attributes
                    ;-1 => End of list
                    ; 0 => Polyline
                    ; 1 => Polymarker
                    ; 2 => Text
                    ; 3 => Filled area
                    ; 4 => None
DEFW            0 for black/white, 1 for colour.
DEFW            0 if text rotation is not possible, 1 if it is.
DEFW            0 if filled areas are not possible, 1 if they are.
DEFW            0 if cannot read cell array, 1 if can.
DEFW            Number of colours in the palette.
                    ; 0 => More than 32767
                    ; 2 => Black and white
DEFW            Number of locator devices (mice, tablets, lightpens)
DEFW            Number of valuator devices (eg: a joystick throttle)
DEFW            Number of choice devices (eg: function keys on a keyboard)
DEFW            Number of string devices (ie: keyboards)
DEFW            Workstation type
                    ; 0 => Output only
                    ; 1 => Input only
                    ; 2 => Input and output
                    ; 3 => Segment storage
                    ; 4 => GEM metafile output.
```

The PTSOUT return array gives:

```
DEFW            minimum character width, minimum character height
DEFW            maximum character width, maximum character height
DEFW            minimum line width,0
DEFW            maximum line width,0
DEFW            minimum marker width, minimum marker height
DEFW            maximum marker width, maximum marker height.
```

(trust marker and character heights rather than widths).

Function 2 - Close workstation

C Interface:

- void v_clswk(WORD handle);

Entered with:

- *function* = 2;
- *handle* = handle to close.

Returns:

- *#intout*=0.

Function 3 - Clear picture

C Interface:

- void v_clrwk(WORD handle);

Entered with:

- *function*=3;

Empties the buffer containing the current picture. This may be the screen, or buffered data for something like a printer.

Function 4 - Output graphics

C Interface:

- void v_updwk(WORD handle);

Entered with:

- *function*=4;

Ensures that all graphics have been displayed which should be displayed. On a device such as a printer, this will cause the picture to be printed.

Function 5 - Escape

Entered with:

- *function*=5;
- *special*=escape number;
- Other values as required by the escape.

Returns: values vary.

Escapes are described in their own listing.

Function 6 - Draw a polyline

C Interface:

- void v_pline(WORD handle, WORD count, WORD xy[]);

Entered with:

- *function*=6;
- *#ptsin*=no. of points in polyline;
- PTSIN contains the coordinates.

Returns: Nothing.

This draws a sequence of lines on the current device. It does not join the first and last point.

Function 7 - Plot a group of markers

C Interface:

- void v_pmarker(WORD handle, WORD count, WORD xy[]);

Entered with:

- *function*=7;
- *#ptsin*=no. of markers;
- PTSIN contains the coordinates.

Returns: Nothing.

Function 8 - Draw text

C Interface:

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

Entered with:

- *function*=8;
- *#ptsin*=1;
- *#intin*=length of text;
- PTSIN contains the coordinates at which to start drawing;
- INTIN contains the text, one character per word.

Returns: Nothing.

Function 9 - Draw a filled polygon.

C Interface:

- void v_fillarea(WORD handle, WORD count, WORD points[]);

Entered with:

- *function*=9;

- *#ptsin*=number of vertices;
- Coordinates of vertices in PTSIN.

Returns: Nothing.

This function does join the first and last points it is given (unlike the polyline).

Function 10 - Output colour index array.

C Interface:

- void v_cellarray(WORD handle, WORD xy[4], WORD row_length, WORD el_per_row, WORD num_rows, WORD wr_mode, WORD *colors);

Entered with:

- *function*=10;
- *#ptsin*=2;
- *#intin*=length of array;
- Array in INTIN;
- Coordinates in PTSIN for bottom left and top right corners;

```
The ptr1 and ptr2 areas of CONTRL hold:
            DEFW    length of each row
            DEFW    number of elements used in each row
            DEFW    number of rows
            DEFW    mode
```

Mode is:

1. replace
2. OR
3. XOR
4. erase

Returns: Nothing.

This function is not implemented in any of the GEM drivers I have seen.

Function 11 - General Drawing Primitive

Entered with:

- *function*=11;
- *special*=primitive ID
- *#ptsin, #intin*, PTSIN, INTIN vary.

General Drawing Primitives are operations such as ellipses and rectangles.

ID=1: void v_bar(WORD handle, WORD xy[4]);
Filled bar. *#ptsin*=2; PTSIN gives diagonally opposite corners.
ID=2: void v_arc(WORD handle, WORD xc, WORD yc, WORD radius, WORD start_angle, WORD end_angle);
Arc. *#ptsin*=4, *#intin*=2. INTIN holds start angle and end angle in 10ths of a degree; PTSIN[0-1] holds coords of centre; PTSIN[6-7] holds (radius, 0). PTSIN[2-5] are zero.

ID=3: void v_pieslice(WORD handle, WORD xc, WORD yc, WORD radius, WORD start_angle, WORD end_angle);
Pie slice. As for arc.
ID=4: WORD v_circle(WORD handle, WORD xc, WORD yc, WORD radius)
Filled circle. *#ptsin*=3; PTSIN[0-1] hold coordinates of the centre; PTSIN[4-5] hold (radius, 0). PTSIN[2-3] hold (0,0).
ID=5: WORD v_ellipse(WORD handle, WORD xc, WORD yc, WORD xrad, WORD yrad);
Draw ellipse. *#ptsin*=2; PTSIN[0-1] hold coordinates of the centre, and PTSIN[2-3] holds (xradius, yradius).
ID=6: WORD v_ellarc(WORD handle, WORD xc, WORD yc, WORD xrad, WORD yrad, WORD sang, WORD eang);
Draw elliptical arc segment. As for the ellipse above, except that INTIN[0] holds start angle and INTIN[1] holds end angle.
ID=7: WORD v_ellpie(WORD handle, WORD xc, WORD yc, WORD xrad, WORD yrad, WORD sang, WORD eang);
Draw elliptical pie slice. Parameters as for the elliptical arc segment.
ID=8: void v_rbox(WORD handle, WORD xy[4]);
Rectangle with rounded corners. *#ptsin*=2; PTSIN gives diagonally opposite corners.
ID=9: void v_rfbox(WORD handle, WORD xy[4]);
Filled rectangle with rounded corners. *#ptsin*=2; PTSIN gives diagonally opposite corners.
ID=10: void v_justified(WORD handle, WORD x, WORD y, BYTE *string, WORD length, WORD word_space, WORD char_space)
Justified text. PTSIN[0-1] = coordinates at which to draw text; PTSIN[2-3] = (length, 0). INTIN[0] holds word spacing and INTIN[1] holds character spacing; INTIN[2] holds the text, one character per word. Thus *#ptsin*=2 and *#intin* = 2 + length of text.
ID=11: void v_etext(WORD handle, WORD x, WORD y, BYTE *string, offsets)
Exact parameters and effects are not known.
ID=12:
Non-filled bezier.
ID=13:
Enable/disable Beziers (v_bez_on() / v_bez_off()).

If *#ptsin* = 0, disable Bezier functions; else enable.

If enabled, this affects Escape 99 and functions 6 and 9.

Function 12 - Set text size

C Interface:

- void vst_height(WORD handle, WORD height, WORD *char_width, WORD *char_height, WORD *cell_width, WORD *cell_height);

Entered with:

- *function*=12;
- *#ptsin*=1
- PTSIN holds (0, height)

Returns:

- PTSOUT[0]=(width,height) of a W

- PTSOUT[1]=(width,height) of a cell

Function 13 - Set text direction

C Interface:

- void vst_rotation(WORD handle, WORD angle);

Entered with:

- *function*=13;
- *#intin*=1
- INTIN[0] = Angle to turn through in tenths of a degree (anticlockwise, 0 = normal)

In practice many drivers only like their text at right-angles.

Function 14 - Set colour index (palette registers)

C Interface:

- void vs_color(WORD handle, WORD colno, WORD *rgb);

Entered with:

- *function*=14;
- *#intin*=4;
- INTIN holds:
    - INTIN[0] = Colour number
    - INTIN[1] = Red 0-1000
    - INTIN[2] = Green 0-1000
    - INTIN[3] = Blue 0-1000

Function 15 - Set line style

C interface:

- WORD vsl_type(WORD handle, WORD style);

Entered with:

- *function*=15;
- *#intin*=1;
- INTIN[0]=style.

Returns:

- INTOUT[0] = style actually selected [The C binding returns this value].

Function 16 - Set line width

C interface:

- WORD vsl_width(WORD handle, WORD width);

Entered with:

- *function*=16;
- *#ptsin*=1;
- PTSIN[0]=(width,0)

Returns:

- PTSOUT[0] = (actual width, 0) [the C binding returns actual width].

Function 17 - Set line colour

C interface:

- WORD vsl_color(WORD handle, WORD colour);

Entered with:

- *function*=17;
- *#intin*=1;
- INTIN[0]=colour.

Returns:

- INTOUT[0] = actual colour [The C binding returns this value].

Function 18 - Set marker type

C interface:

- WORD vsm_type(WORD handle, WORD type);

Entered with:

- *function*=18;
- *#intin*=1;
- INTIN[0]=type.

Returns:

- INTOUT[0] = actual type [The C binding returns this value].

Function 19 - Set marker height

C interface:

- WORD vsm_height WORD handle, WORD height);

Entered with:

- *function*=19;
- *#ptsin*=1;
- PTSIN[0]=(0, height)

Returns:

- PTSOUT[0] = (0, actual height) [The C binding returns actual height].

Function 20 - Set marker colour

C interface:

- WORD vsm_color(WORD handle, WORD colour);

Entered with:

- *function*=20;
- *#intin*=1;
- INTIN[0]=colour.

Returns:

- INTOUT[0] = actual colour [The C binding returns this value].

Function 21 - Set text font

C interface:

- WORD vst_font(WORD handle, WORD font);

Entered with:

- *function*=21;
- *#intin*=1;
- INTIN[0]=font.

Returns:

- INTOUT[0] = font selected [The C binding returns this value].

Function 22 - Set text colour

C interface:

- WORD vst_color(WORD handle, WORD colour);

Entered with:

- *function*=22;
- *#intin*=1;
- INTIN[0]=colour.

Returns:

- INTOUT[0] = actual colour [The C binding returns this value].

Function 23 - Set fill style

C interface:

- WORD vsf_interior(WORD handle, WORD style);

Entered with:

- *function*=23;
- *#intin*=1;
- INTIN[0]=style.

Returns:

- INTOUT[0] = actual style [The C binding returns this value].

Fill style is 0-3: 0=transparent, 1=solid, 2=Pattern, 3=Hatch.

Function 24 - Set fill index

C interface:

- WORD vsf_style(WORD handle, WORD index);

Entered with:

- *function*=24;
- *#intin*=1;
- INTIN[0]=fill index.

Returns:

- INTOUT[0] = actual index [The C binding returns this value].

The fill index is used only with styles 2 & 3. Most GEM drivers support 12 hatches (1-12) and 24 patterns (1-24).

Function 25 - Set fill colour

C interface:

- WORD vsl_color(WORD handle, WORD colour);

Entered with:

- *function*=25;
- *#intin*=1;
- INTIN[0]=colour.

Returns:

- INTOUT[0] = actual colour [The C binding returns this value].

Function 26 - Inquire colour representation (read palette)

C interface:

- WORD vq_color(WORD handle, WORD index, WORD set_flag, WORD rgb[])

Entered with:

- *function*=26;
- *#intin*=2;
- INTIN[0]=colour no.;
    - INTIN[1]=0 to request return values (ie, those set by the program).
    - INTIN[1]=1 to request real values (ie, those actually used by the VDI).

Returns INTOUT[0]=Colour value, INTOUT[1-3]=RGB values 0-1000. The C binding stores the three RGB values in rgb[0-2].

Function 27 - Inquire cell array

C interface:

- WORD vq_cellarray(WORD handle, WORD xy[4], WORD row_len, WORD num_rows, WORD *el_used, WORD *rows_used, WORD *stat, LONG colors)

Entered with:

- *function*=27;
- *#ptsin*=2;
- *#intin*=maximum length of colour index array;
- CONTRL[7]=length of each row;
- CONTRL[8]=number of rows.
- PTSIN holds coordinates of bottom LH and top RH corners of array.

Returns:

- CONTRL[9]=no. elements used in each row. [C stores this in *el_used]
- CONTRL[10]=no. rows used. [C stores this in *rows_used]
- CONTRL[11]=error flag (0=OK 1=error). [C stores this in *stat]
- Array in INTOUT. [C stores this in the array "colors" - the value passed in is a far pointer cast to long].

If INTOUT[x]=-1, the corresponding pixel could not be read.

The next few functions can be operated in Request mode or Sample mode. The information given for them is rather hypothetical, as GEM drivers don't tend to implement these calls (except for the String input, which is used to reaad the keyboard).

Function 28 - Read locator (eg tablet or mouse)

In Request mode:

C interface:

- WORD vrq_locator(WORD handle, WORD initx, WORD inity, WORD *xout, WORD *yout, WORD *term)

Entered with:

- *function*=28;
- *#ptsin*=1;
- *#intin*=1;
- INTIN[0]=locator number (normally 1 for keyboard, 2 for mouse etc.);
- PTSIN gives initial coordinates

Returns:

- PTSOUT=coordinates when key/button pressed [C returns this in *xout, *yout];
- INTOUT[0]=terminator (key pressed, or mouse button +20h); *#intout*=0 if error [C returns this in *term].

In Sample mode:

C interface:

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

Entered with:

- *function*=28;
- *#intin*=1;
- INTIN[0]=locator number (normally 1 for keyboard, 2 for mouse etc.);

Returns:

If coordinates changed:
#ptsout=1 if coordinates changed; new coordinates in PTSOUT[0] [C returns this in *xout, *yout];
If key or button pressed:
#intout=1 if key or button pressed; and key or button in INTOUT[0] [C returns this in *term].

The C function call returns bit 0 set if the coordinates changed, and bit 1 set if a key or button was pressed.

Function 29 - Read valuator

A valuator is a device like a mouse, except that it only operates in one dimension - for example, a volume control or a throttle.

In Request mode:

C interface:

- WORD vrq_valuator(WORD handle, WORD val_in, WORD *val_out, WORD *term)

Entered with:

- *function*=29;
- *#intin*=2;

- INTIN[0]=valuator number.
- INTIN[1]=initial value.

Returns:

- INTOUT[0]=final value when key/button pressed; [C returns this in *val_out]
- INTOUT[1]=terminator (key pressed, or button + 20h); *#intout*=0 if error. [C returns this in *term]

In Sample mode:

C interface:

- WORD vsm_valuator(WORD handle, WORD val_in, WORD *val_out, WORD *term, WORD *status)

Entered with:

- *function*=29;
- *#intin*=1;
- INTIN[0]=valuator number.

Returns:

If value changed:
#intout=1; new value in INTOUT[0].
If key or button pressed:
#intout=2 if key or button pressed; final value in INTOUT[0] and key or button in INTOUT[1].

The C binding returns INTOUT[0] in *val_out, INTOUT[1] in *term, and *#intout* in *status.

Function 30 - Read choice

In Request mode:

C interface:

- WORD vrq_choice(WORD handle, WORD in_choice, WORD *out_choice)

Entered with:

- *function*=30;
- *#intin*=1;
- INTIN[0]=choice number (1=function keys on keyboard).

Returns *#intout*=1, INTOUT[0]=choice (1-n) [C stores this in *out_choice].

In Sample mode:

C interface:

- WORD vsm_choice(WORD handle, WORD *choice)

Entered with:

- *function*=30;
- *#intin*=1;
- INTIN[0]=choice number (1=function keys on keyboard).

Returns:

- *#intout*=0 if nothing happened;
- *#intout*=1 if choice (choice in INTOUT[0]);
- *#intout*=2 if non-choice key (choice in INTOUI[0], key in INTOUT[1]).

The C binding returns *#intout*, and sets *choice to INTOUT[0].

Function 31 - Read string

In Request mode:

C interface:

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

Entered with:

- *function*=31;
- *#ptsin*=INTIN[2];
- *#intin*=3;
- INTIN[0]=device number;
- INTIN[1]=maximum length;
- INTIN[2]=1 to echo, 0 not to.
- If echoing is on, PTSIN[0] gives coordinates of where to echo the string.

Returns *#intout*=length of string returned, string in INTOUT.

The C binding returns the string as series of bytes, 0-terminated, at "string".

In GEM, if the device number is negative, then keyboard scancodes are returned as well as ASCII. In each INTOUT word, the high byte is the scancode and the low byte is the ASCII value.

In Sample mode:

C interface:

- WORD vsm_string(WORD handle, WORD length, WORD echo_mode, WORD echo_xy[], BYTE *string)

Entered with:

- *function*=31;
- *#ptsin*=0;
- *#intin*=2;
- INTIN[0]=device number;
- INTIN[1]=maximum length.

Returns *#intout*=length of string returned, string in INTOUT. The C binding returns a 0-terminated string at "string", and its length as a word.

In GEM, if the device number is negative, then keyboard scancodes are returned as well as ASCII. In each INTOUT word, the high byte is the scancode and the low byte is the ASCII value.

Function 32 - Set writing mode

C interface:

- WORD vswr_mode(WORD handle, WORD mode)

Entered with:

- *function*=32;
- *#intin*=1;
- INTIN[0]=mode.

Note: "background" is the second colour used in dashed lines etc. When such a line is being drawn, the dashes are drawn as "foreground" areas and the gaps as "background" areas. Modes are:

1. Replace. "Foreground" and "background" areas are replaced ("background" areas with colour 0).
2. Transparent. "Foreground" areas are replaced but "background" areas stay the same.
3. XOR. "Foreground" areas are XOR'ed with previous colour; "background" areas stay the same.
4. Erase. "Foreground" areas are written in GSX colour 0; "background" areas stay the same.

Returns the previous mode in INTOUT[0]. [C returns this from the function call].

Function 33 - Set input mode

C interface:

- WORD vsin_mode(WORD handle, WORD dev_type, WORD mode)

Entered with:

- *function*=33;
- *#intin*=2;
- INTIN[0]=device type (1=locator, 2=valuator, 3=choice, 4=string).
- INTIN[1]=mode (1=Request, 2=Sample).

GSX stopped here. Calls from here are only present in the VDI, though the next few calls can be accessed from the GSX-86 interface present in GEM/1 and GEM/2.

Function 34

This function is a no-op in all GEM drivers that I have examined.

Function 35 - Get line attributes

C interface:

- VOID vql_attributes(WORD handle, WORD attributes[])

Entered with:

- *function*=35

Returns:

- INTOUT[0] = Line style
- INTOUT[1] = Line colour
- INTOUT[2] = Drawing mode
- INTOUT[3] = Line start style
- INTOUT[4] = Line end style
- PTSOUT[0] = line width

[The C binding returns these values in `attributes[]`, with line width in `attributes[3]`. You would need to write a modified binding to get at the line start and end styles.]

Function 36 - Get marker attributes

C interface:

- VOID vqm_attributes(WORD handle, WORD attributes[])

Entered with:

- *function*=36

Returns:

- INTOUT[0] = Marker style
- INTOUT[1] = Marker colour
- INTOUT[2] = Drawing mode
- PTSOUT[1] = Marker height

[The C binding returns these values in `attributes[]`, with marker height in `attributes[3]`].

Function 37 - Get fill attributes

C interface:

- VOID vqf_attributes(WORD handle, WORD attributes[])

Entered with:

- *function*=37

Returns:

- INTOUT[0] = Fill style
- INTOUT[1] = Fill index
- INTOUT[2] = Fill colour

- INTOUT[3] = Drawing mode
- INTOUT[4] = Outline flag

[The C binding returns these values in `attributes[]`.]

Function 38 - Get text attributes

C interface:

- VOID vqt_attributes(WORD handle, WORD attributes[])

Entered with:

- *function*=38

Returns:

- INTOUT[0] = Text style
- INTOUT[1] = Text colour
- INTOUT[2] = Text direction
- INTOUT[3] = Justify H mode
- INTOUT[4] = Justify V mode
- INTOUT[5] = Drawing mode
- PTSOUT[1] = Minimum character height
- PTSOUT[3] = Maximum character height

[The C binding returns these values in `attributes[]`, with the minimum and maximum character sizes in `attributes[7]` and `attributes[9]` respectively.]

 The ViewMAX VGA driver does not support this call; you can check for this by seeing if it returned any integers. The normal call returns 6 integers but ViewMAX returns 0. This missing call will cause some programs (eg: RCS.APP) to crash with Divide by Zero errors when the ViewMAX driver is used.

Function 39 - Set text alignment

C interface:

- VOID vst_alignment(WORD handle, WORD hor_in, WORD vert_in, WORD *hor_out, WORD *vert_out);

Entered with:

- *function*=39
- *#intin*=2
- INTIN[0] = horizontal alignment
- INTIN[1] = vertical alignment

Returns:

- INTOUT[0] = actual horizontal alignment
- INTOUT[1] = actual vertical alignment

VDI calls after this point should not be accessed from the GSX-86 interface (INT E0). The VDI interface (INT EF) should be used.

Function 100 - Open virtual workstation

C Interface:

- WORD v_opvnwk(WORD work_in[], WORD *handle, WORD work_out[]);

Entered with:

- *function*=100;
- *#ptsin*=0;
- *#intin*=11;
- *handle*=handle of currently open workstation (returned by function 1).
- INTIN contains initial settings, as in function 1.

Returns:

- *#ptsout*= 6;
- *#intout*= 45;
- *handle* = new handle, 0 if none available.
- INTOUT and PTSOUT contain the device data, as in function 1.

This is used to create a second and subsequent graphics context for an already- open driver. Typically, GEM will open the screen device using function 1, and the applications and desk accessories running under it will use this function to have their own graphics context for the display.

Current VDIs allow eight graphics contexts to be open at a time.

Virtual workstations opened with this call must be closed using function 101.

Function 101 - Close virtual workstation

C Interface:

- void v_clsvwk(WORD handle);

Entered with:

- *function* = 101;
- *handle* = handle to close.

Returns:

- *#intout*=0.

Function 102 - Query extended information

C Interface:

- void vq_extnd(WORD handle, WORD owflag, WORD work_out[]);

Entered with:

- *function* = 102;
- *#intin* = 1;
- INTIN[0] = 0 (read startup information) or 1 (read extra information)

Returns:

- Up to 45 words in INTIN
- Up to 12 words in PTSIN

If INTIN[0] was 0, this returns the same data as function 1. If INTIN[0] was 1, it returns a different set of data:

```
INTOUT: DEFW    Type of alpha/graphic controllers
                        0 => Not a screen
                        1 => Separate text/graphic controllers and
                                separate screens (eg: CGA+MDA dualhead)
                        2 => Separate controllers, common screen
                        3 => One controller, separate screen buffers
                        4 => One controller, one screen buffer
        DEFW    No. of background colours (always 1?)
        DEFW    Text styles supported (0Fh for printers, 1 for screen)
        DEFW    1 if can scale rasters, else 0
        DEFW    No. of planes
        DEFW    1 if palette based, 0 if true colour
        DEFW    Number of 16x16 raster operations per second
        DEFW    1 if v_contourfill() is provided, else 0
        DEFW    Character rotation capability
                        0 => No
                        1 => 90 degree angles only
                        2 => Any angle
        DEFW    No. of writing modes
        DEFW    Highest input mode
                        0 => No input
                        1 => Request only
                        2 => Request and sample
        DEFW    1 if text alignment is possible, else 0
        DEFW    1 if device can ink, else 0
        DEFW    Rubber banding
                        0 => No rubberbanding
                        1 => Lines only
                        2 => Lines and rectangles
        DEFW    Maximum size of PTSIN (in vertices); -1 for unlimited
        DEFW    Max size of INTIN (in words); -1 for unlimited
        DEFW    No. of buttons on mouse
        DEFW    1 if styles are available for wide lines
        DEFW    1 if writing modes are available for wide lines
        DEFW    1 if clip rectangle set, else 0
        DEFW    ?       ;[20] Unknown
        DEFW    ?       ;[21] Unknown
        DEFW    ?       ;[22] Unknown
        DEFW    ?       ;[23] Unknown
        DEFW    ?       ;[24] Unknown
        DEFW    ?       ;[25] Unknown
        DD      Size of offscreen backing buffer. If this is nonzero, then
                the graphics card is assumed to have a buffer outside
                conventional memory, which can be used by vro_cpyfm().
                The GEM 3.x AES supports this buffer mechanism and will
                use it in preference to allocating a buffer in main memory.
;
PTSOUT: DEFW    x,y     ;Top left-hand corner of clip rectangle
        DEFW    x,y     ;Bottom right-hand corner of clip rectangle
```

Function 103 - Contour fill

C Interface:

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

Entered with:

- *function*=103;
- *#ptsin*=1;
- *#intin*=1;
- PTSIN contains the coordinates at which to start drawing;
- INTIN contains the fill index to use.

Returns: Nothing.

This function is not implemented in any of the GEM/3 screen or printer drivers.

Function 104 - Enable/disable fill perimeter

C Interface:

- void vsf_perimeter(WORD handle, WORD per_vis);

Entered with:

- *function*=104;
- *#intin*=1;
- INTIN contains 1 to draw a perimeter round filled areas, 0 not to.

Returns: Nothing.

Function 105 - Get pixel value

C Interface:

- BOOLEAN v_get_pixel(WORD handle, WORD x, WORD y, WORD *pel, WORD *index);

Entered with:

- *function*=105;
- *#ptsin*=1;
- PTSIN contains the point to read.

Returns:

- If successful, *#intout*=2. INTOUT[0] is "pel" and INTOUT[1] is "index".
- If not successful *#intout*=0.

This function is not implemented in any of the GEM/3 screen or printer drivers.

Function 106 - Set text effects

C Interface:

- WORD vst_effects(WORD handle, WORD effect);

Entered with:

- *function*=106;
- *#intin*=1;
- INTIN contains a bitmapped effect:

```
Bit 0: Bold
Bit 1: Light
Bit 2: Skew
Bit 3: Underline
Bit 4: Outline
Bit 5: Shadow
Bits 6-7: Rotate (0=normal, 1=90 degrees, 2=180 degrees, 3=270
                  degrees)
Bit 8: Scale
```

Returns: The actual effect bits that were used.

Function 107 - Set text size in points

C Interface:

- void vst_point(WORD handle, WORD point, WORD *char_width, WORD *char_height, WORD *cell_width, WORD *cell_height);

Entered with:

- *function*=107;
- *#intin*=1
- INTIN holds height in points.

Returns:

- PTSOUT[0]=(width,height) of a W
- PTSOUT[1]=(width,height) of a cell

Function 108 - Set line end style

C Interface:

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

Entered with:

- *function*=108;
- *#intin*=2
- INTIN holds beginning & end style, in that order.

Returns: Actual styles selected in INTOUT.

Styles should be 0 (square), 1 (arrow) or 2 (circle).

Function 109 - Copy raster opaque

C Interface:

- VOID vro_cpyfm(WORD handle, WORD wr_mode, WORD xy[], MFDB *srcMFDB, MFDB *desMFDB);

Entered with:

- *function*=109;
- *#ptsin*=4
- *#intin*=1
- *ptr1* and *ptr2* point at the source and destination MFDBs.
- INTIN holds the writing mode.
- The first two points in PTSIN are opposite corners of a rectangle in the source bitmap; the second two are opposite corners of a rectangle in the destination bitmap.

Returns: Nothing.

This will copy a rectangular area from the source bitmap to the destination bitmap. It should be used for copying between bitmaps with the same colour depth.

Function 110 - Transform bitmap

C Interface:

- VOID vr_trnfm(WORD handle, MFDB *srcMFDB, MFDB *desMFDB);

Entered with:

- *function*=110;
- *ptr1* and *ptr2* point at the source and destination MFDBs.

Returns: Nothing.

This will transform a bitmap from device-independent format to device - dependent, or vice versa.

Function 111 - Set cursor shape

C Interface:

- VOID vsc_form(WORD handle, WORD form[37]);

Entered with:

- *function*=111;
- *#intin*=37;
- INTIN holds a cursor shape:

```
DEFW    hot spot X
DEFW    hot spot Y
DEFW    plct            ;No. of planes in mouse defn. Always 1.
DEFW    ci_mask         ;Colour index for mask
DEFW    ci_data         ;Colour index for pointer
```

```
DEFS    32      ;Mouse pointer mask
DEFS    32      ;Mouse pointer bitmap
```

Returns: Nothing.

Function 112 - Create user-defined pattern

C Interface:

- VOID vsf_udpat(WORD handle, WORD *pattern, WORD planes);

Entered with:

- *function*=112;
- *#intin*=16 * no. of planes in pattern;
- INTIN holds 16 words for each plane in the pattern.

Returns: Nothing.

Function 113 - Create user-defined line style

C Interface:

- VOID vsl_udsty(WORD handle, WORD pattern);

Entered with:

- *function*=113;
- *#intin* = 1;
- INTIN[0] is the pattern to use.

Returns: Nothing.

Function 114 - Draw a filled rectangle

C Interface:

- VOID vr_recfl(WORD handle, WORD *xy);

Entered with:

- *function*=114;
- *#ptsin* = 2;
- PTSIN holds opposite corners of the rectangle.

Returns: Nothing.

No border will be drawn round the rectangle.

Function 115 - Query input mode

C Interface:

- VOID vqin_mode(WORD handle, WORD dev_type, WORD *mode);

Entered with:

- *function*=115;
- *#intin* = 1;
- INTIN[0] = device type (1=locator 2=valuator 3=choice 4=string)

Returns: INTOUT[0] = mode (0=request 1=sample)

Function 116 - Get text extent

C Interface:

- VOID vqt_extent(WORD handle, BYTE *string, WORD *extent);

Entered with:

- *function*=116;
- *#intin* = length of string;
- INTIN contains the string; each word holds one character.

Returns: PTSOUT[0] = (X, Y) size of string.

Function 117 - Get character size

C Interface:

- WORD vqt_width(WORD handle, BYTE character, WORD *cell_width, WORD *left_delta, WORD *right_delta)

Entered with:

- *function*=117;
- *#intin* = 1;
- INTIN[0] = character.

Returns:

- PTSOUT[0] = (cell width, ?)
- PTSOUT[1] = (left delta, ?)
- PTSOUT[2] = (right delta, ?)
- INTOUT[0] = -1 if no bitmap for character in font; else character.

Function 118 - Exchange timer vector

C Interface:

- VOID vex_timv(WORD handle, VOID FAR *tim_addr, VOID FAR *old_addr, WORD *scale);

Entered with:

- *function*=118;
- *ptr1* = address of new timer code.

Returns:

- *ptr2* = address of old timer code
- INTOUT[0] = timer tick count in milliseconds (ie, milliseconds per tick)

On a PC, this sets the INT 0x1C handler.

Function 119 - Load fonts

C Interface:

- VOID vst_load_fonts(WORD handle, WORD select);

Entered with:

- *function*=119.

Returns:

- INTOUT[0] = number of fonts loaded.

The "Select" parameter to the C interface does not appear to be used.

Function 120 - Unload fonts

C Interface:

- VOID vst_unload_fonts(WORD handle, WORD select);

Entered with:

- *function*=120.

Returns: Nothing.

Function 121 - Copy raster transparent

C Interface:

- VOID vrt_cpyfm(WORD handle, WORD wr_mode, WORD xy[], MFDB *srcMFDB, MFDB *desMFDB, WORD fg, WORD bg);

Entered with:

- *function*=121;
- *#ptsin*=4
- *#intin*=3
- *ptr1* and *ptr2* point at the source and destination MFDBs.
- INTIN holds the writing mode, and the foreground and background colours in which the monochrome bitmap is to be painted.
- The first two points in PTSIN are opposite corners of a rectangle in the source bitmap; the second two are opposite corners of a rectangle in the destination bitmap.

Returns: Nothing.

This will copy a rectangular area from the source bitmap to the destination bitmap. It should be used to copy a mono bitmap into a colour bitmap.

Function 122 - Show mouse pointer

C Interface:

- VOID v_show_c (WORD handle, WORD reset)

Entered with:

- *function*=122;
- *#intin*=1;
- INTIN[0] = reset flag.

Returns: Nothing.

If the "reset flag" is nonzero, displays the cursor and sets the "hide count" to 0. If not, decrements the "hide count" and if it is 0, shows the cursor.

Function 123 - Hide mouse pointer

C Interface:

- VOID v_hide_c (WORD handle)

Entered with:

- *function*=123.

Returns: Nothing.

Hides the cursor and increases the "hide count".

Function 124 - Get mouse position

C Interface:

- VOID vq_mouse (WORD handle, WORD *status, WORD *px, WORD *py)

Entered with:

- *function*=124.

Returns:

- INTOUT[0] = mouse button status.
- PTSOUT[0] = mouse cursor coordinates.

Function 125 - Exchange button vector

C Interface:

- VOID vex_butv(WORD handle, VOID FAR *btn_addr, VOID FAR *old_addr);

Entered with:

- *function*=125;
- *ptr1* = address of new button code.

Returns:

- *ptr2* = address of old button code.

The button code will be called when a mouse button is clicked. The mouse button status is passed to it in AX (x86) / D0 (680x0); Bit 0 for the left button, bit 1 for the right, bit 2 for the middle.

Function 126 - Exchange motion vector

C Interface:

- VOID vex_motv(WORD handle, VOID FAR *mot_addr, VOID FAR *old_addr);

Entered with:

- *function*=126;
- *ptr1* = address of new mouse movement code.

Returns:

- *ptr2* = address of old mouse movement code.

The code will be called when the mouse pointer is moved. It is entered with BX = mouse X-coordinate and CX = mouse Y-coordinate (D0 and D1 for 680x0).

Function 127 - Exchange cursor draw vector

C Interface:

- VOID vex_curv(WORD handle, VOID FAR *cur_addr, VOID FAR *old_addr);

Entered with:

- *function*=127;
- *ptr1* = address of new mouse pointer draw code.

Returns:

- *ptr2* = address of old mouse pointer draw code.

The code is entered with BX = mouse X-coordinate and CX = mouse Y-coordinate (D0 and D1 for 680x0).

Function 128 - Get keyboard shift state

C Interface:

- VOID vq_key_s(WORD handle, WORD *status);

Entered with:

- *function*=128.

Returns:

- INTOUT[0] = status (bit 0 = Shift, bit 1 = Ctrl, bit 2 = Alt)

Function 129 - Set clipping rectangle

C Interface:

- VOID vs_clip(WORD handle, WORD clip_flag, WORD xy[]);

Entered with:

- *function*=129;
- *#ptsin*=2;
- *#intin*=1;
- PTSIN holds coordinates of opposite corners of the rectangle;
- INTIN[0] is 0 to turn off clipping, 1 to turn it on.

Returns: Nothing.

Function 130 - Get font name

C Interface:

- WORD vqt_name(WORD handle, WORD element_num, BYTE name[])

Entered with:

- *function*=130;
- *#intin*=1;
- INTIN[0] is index of font (1=first, 2=second...)

Returns:

- *#intout*=33;
- INTOUT[0] = font ID (eg: 2 for serif, 14 for sans-serif)
- INTOUT[1-32] = font name, each word is one character.

 In GEM/5, this function returns nothing for font index 0 or 1; for other indices, the font scaler is used to return data. A good test for a GEM/5 VDI would be to do a `vqt_name()` on font 1, and see if it returned any values - if it did not, GEM/5 is running.

Function 131 - Get font info

C Interface:

- BOOLEAN vqt_font_info(WORD handle, WORD *minADE, WORD *maxADE, WORD distances[], WORD *maxwidth, WORD effects[])

Entered with:

- *function*=131.

Returns:

- *#intout*=2 if successful, 0 if failed;
- *#ptsout*=5 if successful, 0 if failed;
- INTOUT[0] = first character in font (minADE);
- INTOUT[1] = last character in font (maxADE;
- PTSOUT[0] = (max cell width, bottom);
- PTSOUT[1] = (weight, descent);
- PTSOUT[2] = (left offset, half);
- PTSOUT[3] = (right offset, ascent);
- PTSOUT[4] = (?, top).

This returns information for the currently selected font.

"weight" (in PTSOUT[1]) is 0 if the font is not monospaced.

"left offset" and "right offset" are the amounts the character slants when it is skewed.

The C binding returns distances[] = {bottom, descent, half, ascent, top} and effects[] = {weight, left offset, right offset}.

Function 132 - Justified text

C Interface:

- WORD vqt_justified(WORD handle, WORD x, WORD y, BYTE string[], WORD length, WORD word_space, WORD char_space, WORD *offsets)

Entered with:

- *function*=132;
- *#ptsin*=2;
- *#intin*=2 + no. of bytes in string;
- PTSIN[0] = (x, y) to draw string at;
- PTSIN[1] = (width, 0) of area string should occupy;
- INTIN[0] = word space;
- INTIN[1] = character space;
- INTIN[2..] = string, one byte in each word.

Returns:

- *#ptsout*=?;
- PTSOUT = ?

Functions 133-135

These functions exist in the GEM/4 screen driver, but have no effect.

Function 136

GEM/4: Based on v_pline() but takes extra integer parameters.

Function 137

GEM/4: Based on v_fillarea() but takes extra integer parameters.

Function 142

GEM/4: Takes an integer parameter.

Function 143

GEM/4: Takes an integer parameter that is at most 100.

Function -1 - GDOS escape

These are functions handled by the GDOS, where no driver handle is required.

Entered with:

- *function*=-1;
- *special*= subfunction.

 GEM/2 only provides subfunction 1.

Subfunctions are

1. Page out GEM to run a DOS (or CP/M-86) program. Only a stub of the VDI will remain in memory, and no VDI or AES functions will be available.
   Entered with INTIN[0-1] = far pointer to INT 21/AX=4B00 parameter block and INTIN[2-3] = address of 0-terminated error message.
2. Font extension. Entered with 3 words in INTIN, which are the file type to use for fonts (normally 'F','N','T').
3. PD list. Returns path to GDOS in INTOUT and driver paths at *ptr1*.
4. Driver list. Enter with INTIN[0] = device ID and INTIN[1] = information requested:
   1. Full driver filename including path
   2. Short driver description
   3. Long driver description
   4. Driver font search path
   5. Driver patch bytes

   C interface: v_get_driver_info(WORD device_id, WORD info_select, BYTE *info_string);

   Returns information in INTOUT, and *#intout* set accordingly.
5. Get font info. Returns INTOUT[0] = segment of font buffer and INTOUT[1] = size of font buffer in bytes.
6. Set App buffer address and size. Entered with INTIN[0-1] = far pointer to buffer and INTIN[2] = buffer size in paragraphs. Used in GEM/3 for beziers.

7. (GEM/4) Fill INTOUT with a 0-terminated ASCII string.
8. (GEM/4) Fill INTOUT with a 0-terminated ASCII version number (for GEM/4, this is "3.91".

This text was originally created by John Elliott, and was located on his website www.seasip.info.
This version of the document was packaged by Shane M. Coughlan for the OpenGEM SDK.