

VDI/GDOS Function Reference

v_alpha_text()

VOID v_alpha_text(*handle*, *str*)
 WORD *handle*;
 char **str*;

v_alpha_text() outputs a line of alpha text.

OPCODE 5

SUB-OPCODE 25

AVAILABILITY Supported by all printer and metafile drivers.

PARAMETERS *handle* is a valid workstation handle. *str* is a pointer to a null-terminated text string which will be printed. Two special **BYTE** codes may be embedded in the text. ASCII 12 will cause a printer form-feed. ASCII 18 'DC2' will initiate an escape sequence followed by a command descriptor **BYTE** (in ASCII) indicating which action to take as follows.

Command BYTE	Meaning
'0'	Enable bold print.
'1'	Disable bold print.
'2'	Enable italic print.
'3'	Disable italic print.
'4'	Enable underlining.
'5'	Disable underlining.
'6'	Enable superscript.
'7'	Disable superscript.
'8'	Enable subscript.
'9'	Disable subscript.
'A'	Enable NLQ mode.
'B'	Disable NLQ mode.
'C'	Enable wide printing.
'D'	Disable wide printing.
'E'	Enable light printing.
'F'	Disable light printing.
'W'	Switch to 10-cpi printing.
'X'	Switch to 12-cpi printing.
'Y'	Toggle compressed printing.
'Z'	Toggle proportional printing.

BINDING WORD i = 0;

```
while(intin[i++] = (WORD)*str++);
```

```
contrl[0] = 5;
```

```
contrl[1] = 0;
```

```
contrl[3] = --i;
```

```
contrl[5] = 25;
```

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

vdi();

CAVEATS The line of text must not exceed the maximum allowable length of the *intin* array as returned by **vq_extnd()** or the maximum length of your compilers' array.

COMMENTS Only commands '0', '1', '2', '3', '4', and '5' are available with most printer drivers.

SEE ALSO **v_gtext(), v_ftext()**

v_arc()

VOID v_arc(*handle, x, y, radius, startangle, endangle*)

WORD *handle, x, y, radius, startangle, endangle*;

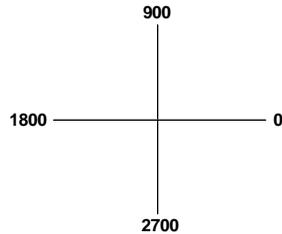
v_arc() outputs an arc to the specified workstation.

OPCODE 11

SUB-OPCODE 2

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by **v_opnvwk()** or **v_opnwk()**.

PARAMETERS *handle* is a valid workstation handle. *x* and *y* specify the center of an arc with a radius of *radius* and starting and ending angles of *startangle* and *endangle* specified in tenths of degrees as follows:



BINDING

```

contrl[0] = 11;
contrl[1] = 4;
contrl[3] = contrl[5] = 2;
contrl[6] = handle;

intin[0] = startangle;
intin[1] = endangle;

ptsin[0] = x;
ptsin[1] = y;
ptsin[2] = ptsin[3] = ptsin[4] = ptsin[5] = 0;
ptsin[6] = radius;
ptsin[7] = 0;

vdi();

```

SEE ALSO `vsl_color()`

v_bar()

VOID v_bar(*handle*, *pxy*)

WORD *handle*;

WORD **pxy*;

v_bar() outputs a filled rectangle to the specified workstation.

OPCODE 11

SUB-OPCODE 1

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by `v_opnvwk()` or `v_opnwk()`.

PARAMETERS	<i>handle</i> is a valid workstation handle. <i>pxy</i> points to an array of four WORD s specifying a VDI format rectangle to output.
BINDING	<pre>contrl[0] = 11; contrl[1] = 2; contrl[3] = 0; contrl[5] = 1; contrl[6] = handle; ptsin[0] = pxy[0]; ptsin[1] = pxy[1]; ptsin[2] = pxy[2]; ptsin[3] = pxy[3]; vdi();</pre>
COMMENTS	This function, as opposed to vr_rectfl() , <i>does</i> take the setting of vsf_perimeter() into consideration.
SEE ALSO	vsf_interior() , vsf_style() , vsf_color() , vsf_perimeter() , vsf_udpat()

v_bez()

VOID v_bez(*handle*, *count*, *pxy*, *bezarr*, *extent*, *totpts*, *totmoves*)

WORD *handle*, *count*;

WORD **pxy*, **extent*;

char **bezarr*;

WORD **totpts*, **totmoves*;

v_bez() outputs a bezier curve path.

OPCODE 6

SUB-OPCODE 13

AVAILABILITY Available only with **FONTGDOS**, **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* is a valid workstation handle. *count* specifies the number of vertices in the path. *pxy* is a pointer to a **WORD** array (*count* * 2) **WORD**s long containing the vertices where *pxy[0]* is the X coordinate of the first point, *pxy[1]* is the Y coordinate of the first point and so on. *bezarr* is a pointer to a character array *count* **BYTE**s long where each byte is a bit mask with two flags that dictate the interpretation of each vertice as follows:

Name	Bit	Meaning
BEZ_BEZIER (0x01) BEZ_POLYLINE (0x00)	0	If set, begin a 4-point bezier segment (two anchor points followed by two control points), otherwise, begin a polyline segment.
BEZ_NODRAW (0x02)	1	If set, jump to this point without drawing.
—	2-7	Currently unused (set to 0).

Upon exit, a 4 **WORD** array pointed to by *extent* is filled in with a **VDI** format rectangle defining a bounding box of the path drawn. The **WORD** pointed to by *totpts* is filled in with the number of points in the resulting path whereas the total number of moves is stored in the **WORD** pointed to by *totmoves*.

BINDING

```
WORD i;

contrl[0] = 6;
contrl[1] = count;
contrl[3] = (count + 1)/2;
contrl[5] = 13;
contrl[6] = handle;

for(i = 0; i < count; i++)
{
    intin[i] = (WORD)bezarr[i];
    ptsin[ i*2 ] = pxy[ i*2 ];
    ptsin[ (i*2) + 1 ] = pxy[ (i*2) + 1];
}

vdi();

*totpts = intin[0];
*totmoves = intin[1];

for(i = 0; i < 4; i++)
    extent[i] = ptsout[i];
```

SEE ALSO

v_bez_fill(), v_bez_on(), v_bez_off(), v_bez_qual(), v_set_app_buff()

v_bez_fill()

VOID v_bez_fill(*handle*, *count*, *pxy*, *bezarr*, *extent*, *totpts*, *totmoves*)

WORD *handle*, *count*;

WORD **pxy*, **extent*;

char **bezarr*;

WORD **totpts*, **totmoves*;

v_bez_fill() outputs a filled bezier path.

OPCODE

9

SUB-OPCODE	13
AVAILABILITY	Available only with FONTGDOS , FSMGDOS or SpeedoGDOS .
PARAMETERS	Same as v_bez() .
BINDING	<pre>WORD i; contrl[0] = 9; contrl[1] = count; contrl[3] = (count + 1)/2; contrl[5] = 13; contrl[6] = handle; for(i = 0; i < count * 2; i++) ptsin[i] = pxy[i]; for(i = 0; i < count; i++) intin[i] = (WORD)bezarr[i]; vdi(); *totpts = intin[0]; *totmoves = intin[1]; for(i = 0; i < 4; i++) extent[i] = ptsout[i];</pre>
SEE ALSO	v_bez() , v_bez_on() , v_bez_off() , v_bez_qual() , v_set_app_buff()

v__bez__off()

VOID **v_bez_off**(*handle*)

WORD *handle*;

v_bez_off() disables bezier capabilities and frees associated memory.

OPCODE 11

SUB-OPCODE 13

AVAILABILITY Available only with **FONTGDOS**, **FSM**, or **SpeedoGDOS**.

PARAMETERS *handle* is a valid workstation handle.

BINDING

```
contrl[0] = 11;
contrl[1] = 0;
contrl[3] = 0;
contrl[5] = 13;
```

```
    contrl[6] = handle;  
  
    vdi();
```

COMMENTS This function should be called to free any memory reserved by the bezier functions.

SEE ALSO v_bez_on()

v_bez_on()

WORD v_bez_on(*handle*)

WORD *handle*;

v_bez_on() enables bezier capabilities.

OPCODE 11

SUB-OPCODE 13

AVAILABILITY Available only with **FONTGDOS**, **FSM**, or **SpeedoGDOS**.

PARAMETERS *handle* is a valid workstation handle.

BINDING

```
    contrl[0] = 11;  
    contrl[1] = 1;  
    contrl[3] = 0;  
    contrl[5] = 13;  
    contrl[6] = handle;  
  
    vdi();  
  
    return intout[0];
```

RETURN VALUE v_bez_on() returns a **WORD** value indicating the number of line segments each curve is composed of (smoothness). The value returned (0-7) is a power of 2 meaning that a return value of 7 indicates 128 line segments per curve.

SEE ALSO v_bez_off()

v_bez_qual()

VOID v_bez_qual(*handle*, *percent*, *actual*)

WORD *handle*, *percent*;

WORD **actual*;

v_bez_qual() sets the speed/quality ratio of the bezier curve rendering engine.

OPCODE

5

SUB-OPCODE

99

AVAILABILITY

Available only with **FONTGDOS**, **FSM**, or **SpeedoGDOS**.

PARAMETERS

handle specifies a valid workstation handle. *percent* is a value (0–100) specifying the tradeoff between bezier quality and speed. A value of 0 renders a bezier fastest with the lowest quality while a value of 100 renders a bezier slowest with the highest possible quality. On return, the **WORD** pointed to by *actual* will contain the actual value used.

BINDING

```
contrl[0] = 5;  
contrl[1] = 0;  
contrl[3] = 3;  
contrl[5] = 99;  
contrl[6] = handle;
```

```
intin[0] = 32;  
intin[1] = 1;  
intin[2] = percent;
```

```
vdi();
```

```
*actual = intout[0];
```

COMMENTS

actual may not be an exact percentage as the rendering engine may not actually support every value possible between 1–99.

SEE ALSO

v_bez(), v_bez_fill(), v_bez_on()

v_bit_image()

VOID v_bit_image(*handle*, *fname*, *ratio*, *xscale*, *yscale*, *halign*, *valign*, *pxy*)

WORD *handle*;

char **fname*;

WORD *aspect*, *xscale*, *yscale*, *halign*, *valign*;

WORD **pxy*;

v_bit_image() outputs a disk-based **GEM** '.IMG' file.

OPCODE 5

SUB-OPCODE 23

AVAILABILITY Supported by all printer, metafile, and memory drivers.

PARAMETERS *handle* is a valid workstation handle. *fname* specifies the **GEMDOS** file specification for the **GEM** bit-image file to print. *ratio* should be 0 to ignore the aspect ratio of the image and 1 to adhere to it.

xscale and *yscale* specify the method of scaling to apply to the image. Fractional scaling is specified by a value of 0 whereas a value of 1 represents integer scaling.

If fractional scaling is used, the image will be displayed at the coordinates given by the **VDI** format rectangle pointed to by *pxy*. If integer scaling is applied, the image will be displayed as large as possible within the given coordinates using *halign* and *valign* to specify the image justification as follows:

Value	<i>halign</i>	<i>valign</i>
0	Left IMAGE_LEFT	Top IMAGE_TOP
1	Center IMAGE_CENTER	Center IMAGE_CENTER
2	Right IMAGE_RIGHT	Bottom IMAGE_BOTTOM

BINDING

```
WORD tmp = 5;

intin[0] = ratio;
intin[1] = xscale;
intin[2] = yscale;
intin[3] = halign;
intin[4] = valign;
while(intin[tmp++] = (WORD)*fname++);

contrl[0] = 5;
```

```
contrl[1] = 2;
contrl[3] = --tmp;
contrl[5] = 23;
contrl[6] = handle;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];

vdi();
```

COMMENTS A flag indicating whether the device supports scaling can be found in `vq_extnd()`. This call used with the memory driver can provide image scaling for transfer to the screen with `vrt_cpyfm()`.

SEE ALSO `vq_scan()`

v_cellarray()

VOID `v_cellarray(handle, pxy, rowlen, elements, num_rows, wrmode, colarray)`

WORD `handle`;

WORD `*pxy`;

WORD `rowlen, elements, num_rows, wrmode`;

WORD `*colarray`;

`v_cellarray()` outputs an array of colored cells.

OPCODE 10

AVAILABILITY Not supported by any current drivers.

PARAMETERS `handle` specifies a valid workstation handle. `pxy` points to a **WORD** array with 4 entries specifying a **VDI** format rectangle giving the extent of the array to output.

`rowlen` specifies the length of each color array row. `elements` specifies the total number of color array elements. `num_rows` specifies the number of rows in the color array. `wrmode` specifies a valid writing mode (1–4) and `colarray` points to an array of **WORDS** (`num_rows * elements`) long.

BINDING `WORD i`;

```
contrl[0] = 10;
contrl[1] = 2;
contrl[3] = num_rows * elements;
contrl[6] = handle;
contrl[7] = rowlen;
contrl[8] = elements;
contrl[9] = num_rows;
```

```

    contrl[10] = wrt_mode;

    for(i = 0;i < (num_rows * elements);i++)
        intin[i] = colarray;

    ptsin[0] = pxy[0];
    ptsin[1] = pxy[1];
    ptsin[2] = pxy[2];
    ptsin[3] = pxy[3];

    vdi();

```

CAVEATS This function is not guaranteed available in any driver and should therefore be avoided unless you are sure the driver you are utilizing understands it.

SEE ALSO [vq_cellarray\(\)](#)

v_circle()

VOID v_circle(*handle*, *x*, *y*, *radius*)

WORD *handle*, *x*, *y*, *radius*;

v_circle() outputs a filled circle.

OPCODE 11

SUB-OPCODE 4

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by [v_opnvwk\(\)](#) or [v_opnwk\(\)](#).

PARAMETERS *handle* specifies a valid workstation. *x* and *y* specify the center of a circle with a radius of *radius*.

BINDING

```

    contrl[0] = 11;
    contrl[1] = 3;
    contrl[3] = 0;
    contrl[5] = 4;
    contrl[6] = handle;

    ptsin[0] = x;
    ptsin[1] = y;
    ptsin[2] = ptsin[3] = 0;

    vdi();

```

SEE ALSO [vsf_color\(\)](#), [vsf_interior\(\)](#), [vsf_style\(\)](#), [vsf_udpat\(\)](#)

v_clear_disp_list()

VOID v_clear_disp_list(*handle*)

WORD *handle*;

v_clear_disp_list() clears the display list of a workstation.

OPCODE 5

SUB-OPCODE 22

AVAILABILITY Supported by printer, plotter, metafile, and camera drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 22;  
contrl[6] = handle;  
  
vdi();
```

COMMENTS v_clear_disp_list() is essentially the same as v_clrwk() except that no form feed is issued.

SEE ALSO v_clrwk()

v_clrwk()

VOID v_clrwk(*handle*)

WORD *handle*;

v_clrwk() clears a physical workstation.

OPCODE 3

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation.

BINDING

```
contrl[0] = 3;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;
```

```
vdi();
```

COMMENTS Physical workstations are cleared automatically when they are opened.

This call will generate a form feed on page-oriented devices.

Using this command on a virtual workstation will clear the underlying physical workstation. This is generally not recommended because it will effect all virtual workstations not simply your own.

SEE ALSO `v_clear_disp_list()`, `v_updvwk()`

v_clsvwk()

VOID `v_clsvwk(handle)`

WORD `handle`;

`v_clsvwk()` closes a virtual workstation.

OPCODE 101

AVAILABILITY Supported by all drivers.

PARAMETERS `handle` specifies a valid virtual workstation to close.

BINDING

```
contrl[0] = 101;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;
```

```
vdi();
```

SEE ALSO `v_opnvwk()`

v_clsawk()

VOID `v_clsawk(handle)`

WORD `handle`;

`v_clsawk()` closes a physical workstation.

OPCODE 2

AVAILABILITY Available only with some form of **GDOS**.

PARAMETERS *handle* specifies a valid physical workstation to close.

BINDING

```
contrl[0] = 2;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;  
  
vdi();
```

SEE ALSO `v_opnvwk()`

v_contourfill()

VOID `v_contourfill(handle, x, y, color)`

WORD *handle, x, y, color*;

`v_countourfill()` outputs a ‘seed’ fill.

OPCODE 103

AVAILABILITY Supported by all *current* screen, printer and metafile drivers. The availability of this call can be checked for using `vq_extnd()`.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the starting point for the fill. If *color* is **OTHER_COLOR** (-1) then the fill continues in all directions until a color other than that found in *x* and *y* is found. If *color* is positive then the fill continues in all directions until color *color* is found.

BINDING

```
contrl[0] = 103;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;  
  
intin[0] = color;  
  
ptsin[0] = x;  
ptsin[1] = y;  
  
vdi();
```

COMMENTS In true-color mode if a positive value for *color* is used, the fill spreads until a pixel is found with the same color as ‘virtual pen’ *color*.

SEE ALSO `vsf_color()`, `vsf_interior()`, `vsf_style()`, `vsf_udpat()`

v_curdown()

VOID v_curdown(*handle*)

WORD *handle*;

v_curdown() moves the text cursor down one line.

OPCODE 5

SUB-OPCODE 5

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 5;
contrl[6] = handle;
```

vdi();

COMMENTS This call is equivalent to the ESC-B VT-52 code.

SEE ALSO v_curup()

v_curhome()

VOID v_curdown(*handle*)

WORD *handle*;

v_curhome() moves the text cursor to the upper-left of the screen.

OPCODE 5

SUB-OPCODE 8

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 8;
```

```
    contrl[6] = handle;  
    vdi();
```

COMMENTS This call is equivalent to the ESC-H VT-52 code.

v_curleft()

VOID v_curleft(*handle*)

WORD *handle*;

v_curleft() moves the text cursor left one character position.

OPCODE 5

SUB-OPCODE 7

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* is a valid workstation handle.

BINDING

```
    contrl[0] = 5;  
    contrl[1] = contrl[3] = 0;  
    contrl[5] = 7;  
    contrl[6] = handle;  
  
    vdi();
```

COMMENTS This call is equivalent to the ESC-D VT-52 code.

SEE ALSO v_currigh()

v_currigh()

VOID v_currigh(*handle*)

WORD *handle*;

v_currigh() moves the text cursor one position to the right.

OPCODE 5

SUB-OPCODE 6

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```

contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 6;
contrl[6] = handle;

vdi();

```

COMMENTS This call is equivalent to the ESC-C VT-52 code.

SEE ALSO v_curleft()

v_curtext()

VOID v_curtext(*handle*, *str*)

WORD *handle*;

char **str*;

v_curtext() outputs a line of text to the screen in text mode.

OPCODE 5

SUB-OPCODE 12

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* is a valid workstation handle. *str* is a character pointer to a string no more than 127 characters long.

BINDING

```

WORD i = 0;

while(intin[i++] = (WORD)*str++);

intin[i] = 0;
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = --i;
contrl[5] = 12;
contrl[6] = handle;

vdi();

```

COMMENTS The line of text must not exceed the maximum length of the intin array as returned by vq_extnd() or the maximum length of your compilers' array.

SEE ALSO vs_curaddress(), v_rvon(), v_rvoff()

v_curup()

VOID v_curup(*handle*)

WORD *handle*;

v_curup() moves the text cursor up one line.

OPCODE 5

SUB-OPCODE 4

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 4;  
contrl[6] = handle;  
  
vdi();
```

COMMENTS This call is equivalent to the ESC-A VT-52 code.

SEE ALSO v_curdown()

v_dspcur()

VOID v_dspcur(*handle*, *x*, *y*)

WORD *handle*, *x*, *y*;

v_dspcur() displays the mouse pointer on screen at the specified position.

OPCODE 5

SUB-OPCODE 18

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the screen coordinates of where to display the mouse pointer.

BINDING

```
contrl[0] = 5;  
contrl[1] = 1;  
contrl[3] = 0;  
contrl[5] = 18;  
contrl[6] = handle;  
  
ptsin[0] = x;  
ptsin[1] = y;  
  
vdi();
```

COMMENTS This call will render a mouse cursor on screen regardless of its current ‘show’ status. Normally a function will use either **graf_mouse()** if using the **AES** or **v_show_c()** if using the **VDI**.

SEE ALSO **v_rmcur()**, **graf_mouse()**, **v_show_c()**

v_eeol()

VOID v_eeol(*handle*)

WORD *handle*;

v_eeol() erases the text line from the current cursor position rightwards.

OPCODE 5

SUB-OPCODE 10

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 10;  
contrl[6] = handle;  
  
vdi();
```

COMMENTS This call is equivalent to the ESC-K VT-52 code.

SEE ALSO **v_eeos()**

v_eeos()

WORD v_eeos(*handle*)

WORD *handle*;

v_eeos() erases the current screen of text from the cursor position.

OPCODE

5

SUB-OPCODE

9

AVAILABILITY

Supported by all screen drivers.

PARAMETERS

handle specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 9;  
contrl[6] = handle;
```

```
vdi();
```

COMMENTS

This call is equivalent to the ESC-J VT-52 code.

SEE ALSO

v_eeol()

v_ellarc()

VOID v_ellarc(*handle*, *x*, *y*, *xradius*, *yradius*, *startangle*, *endangle*)

WORD *handle*, *x*, *y*, *xradius*, *yradius*, *startangle*, *endangle*;

v_ellarc() outputs an elliptical arc segment.

OPCODE

11

SUB-OPCODE

6

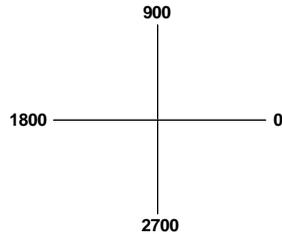
AVAILABILITY

Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

PARAMETERS

handle specifies a valid workstation handle. *x* and *y* specify the coordinates of the

center of an arc with an X radius of *xradius* and a Y radius of *yradius*. Only the portion of the arc which falls between the angles specified in *startangle* and *endangle* will be drawn. Angles are specified in tenths of degrees as follows:

**BINDING**

```

contrl[0] = 11;
contrl[1] = contrl[3] = 2;
contrl[5] = 6;
contrl[6] = handle;

intin[0] = startangle;
intin[1] = endangle;

ptsin[0] = x;
ptsin[1] = y;
ptsin[2] = xradius;
ptsin[3] = yradius;

vdi();

```

SEE ALSO

v_ellipse(), v_ellpic(), vsl_color(), vsl_type(), vsl_width(), vsl_udsty()

v_ellipse()

VOID v_ellipse(*handle*, *x*, *y*, *xradius*, *yradius*)

WORD *handle*, *x*, *y*, *xradius*, *yradius*;

v_ellipse() outputs a filled ellipse.

OPCODE 11

SUB-OPCODE 5

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk().

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the center point of an arc with an X radius of *xradius* and a Y radius of *yradius*.

BINDING

```
contrl[0] = 11;
contrl[1] = 2;
contrl[3] = 0;
contrl[5] = 5;
contrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;
ptsin[2] = xradius;
ptsin[3] = yradius;

vdi();
```

SEE ALSO `v_ellipse()`, `v_ellarc()`, `vsf_color()`, `vsf_interior()`, `vsf_style()`, `vsf_udpat()`, `vs_perimeter()`

v_ellipse()

VOID `v_ellipse(handle, x, y, xradius, yradius, startangle, endangle)`

WORD `handle, x, y, xradius, yradius, startangle, endangle;`

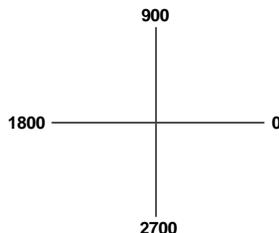
`v_ellipse()` outputs a filled elliptical pie segment.

OPCODE 11

SUB-OPCODE 7

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by `v_opnvwk()` or `v_opnwk()`.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the center coordinates of an elliptical pie segment to draw with an X radius of *xradius* and a Y radius of *yradius*. Only the portion of the arc will be drawn falling between the angles specified in *startangle* and *endangle* (as shown below). The ends of this arc is connected to the center point with lines forming the pie segment.



BINDING

```

contrl[0] = 11;
contrl[1] = contrl[3] = 2;
contrl[5] = 7;
contrl[6] = handle;

intin[0] = startangle;
intin[1] = endangle;

ptsin[0] = x;
ptsin[1] = y;
ptsin[2] = xradius;
ptsin[3] = yradius;

vdi();

```

SEE ALSO `v_ellarc()`, `v_ellipse()`, `vsf_color()`, `vsf_style()`, `vsf_interior()`, `vsf_udpat()`, `vs_perimeter()`

v_enter_cur()

VOID `v_enter_cur(handle)`

WORD *handle*;

`v_enter_cur()` clears the screen to color 0, removes the mouse cursor and enters text mode.

OPCODE 5

SUB-OPCODE 3

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```

contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 3;
contrl[6] = handle;

vdi();

```

CAVEATS You should check that the left mouse button has been released with `vq_mouse()` prior to calling this function. If the button is depressed when you call this function the **VDI** will lock waiting for it to be released after `v_exit_cur()`.

COMMENTS This call is used by a **GEM** application to prepare for executing a **TOS** application when not running under **MultiTOS**.

SEE ALSO `v_exit_cur()`

v_exit_cur()

VOID `v_exit_cur(handle)`

WORD `handle`;

`v_exit_cur()` exits text mode and restores the mouse pointer.

OPCODE 5

SUB-OPCODE 2

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 2;  
contrl[6] = handle;
```

`vdi()`;

CAVEATS See `v_enter_cur()`.

COMMENTS To completely restore the screen you should call `form_dial(FMD_FINISH, sx, sy, sw, sh)` where *sx*, *sy*, *sw*, and *sh* are the coordinates of the screen.

SEE ALSO `v_enter_cur()`

v_fillarea()

VOID `v_fillarea(handle, count, pxy)`

WORD `handle, count`;

WORD `*pxy`;

`v_fillarea()` outputs a filled polygon.

OPCODE 9

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *count* specifies the number of vertices in the polygon to output. *pxy* should point to an array of coordinate pairs with the first **WORD** being the first X point, the second **WORD** being the first Y point and so on.

BINDING

```
WORD i;  
  
contrl[0] = 9;  
contrl[1] = count;  
contrl[3] = 0;  
contrl[6] = handle;  
  
for(i = 0;i < count*2;i++)  
    ptsin[i] = pxy[i];  
  
vdi();
```

COMMENTS This function will automatically connect the first point with the last point.

SEE ALSO **v_pline()**, **v_contourfill()**

v_flushcache()

VOID **v_flushcache()** (*handle*)
WORD *handle*;

v_flushcache() flushes the character bitmap portion of the cache.

OPCODE 251

AVAILABILITY Available only with **FSMGDOS** and **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 251;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;  
  
vdi();
```

SEE ALSO **v_loadcache()**, **v_savecache()**

v_fontinit()

VOID v_fontinit(*fptr_high*, *fptr_low*)

WORD *fptr_high*, *fptr_low*;

v_fontinit() allows replacement of the built-in system font.

OPCODE

5

SUB-OPCODE

102

AVAILABILITY

All TOS versions.

PARAMETERS

fptr_high and *fptr_low* are the high and low **WORDS** of a pointer to a Line-A compatible font header structure in Motorola (Big-Endian) format which contains information about the font to be used as a replacement for the system font.

BINDING

```
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 102;
contrl[6] = handle;

intin[0] = fptr_high;
intin[1] = fptr_low;

vdi();
```

COMMENTS

This function has never been officially documented though it exists in all current versions of TOS.

v_form_adv()

VOID v_form_adv(*handle*)

WORD *handle*;

v_form_adv() outputs the current page without clearing the display list.

OPCODE

5

SUB-OPCODE

20

AVAILABILITY

Supported by all drivers.

PARAMETERS

handle specifies a valid workstation handle.

BINDING

```

contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 20;
contrl[6] = handle;

vdi();

```

COMMENTS This function is useful if you wish to print a new page containing the same objects as on the previous page.

SEE ALSO v_updwk()

v_ftext()

VOID v_ftext(*handle*, *x*, *y*, *str*)

WORD *handle*, *x*, *y*;

char **str*;

v_ftext() outputs outline text taking spacing remainders into consideration.

OPCODE 241

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the starting coordinate of the **NULL**-terminated text string (see **vst_alignment()**) pointed to by *str* to print.

BINDING

```

WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 241;
contrl[1] = 1;
contrl[3] = --i;
contrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;

vdi();

```

COMMENTS The text contained in *str* (including its **NULL** byte) should not exceed the maximum allowable size of the *intin* array (as indicated in the *work_out* array) or the size of the *intin* array allocated by your compiler.

To output 16-bit Speedo character indexes, use **v_ftext16()**.

This function produces output more properly spaced than with `v_gtext()` because it takes the remainder amounts from `vqt_f_extent()` into consideration.

SEE ALSO `v_text()`, `v_text_offset()`, `v_text_offset16()`, `v_gtext()`, `vst_alignment()`, `vst_color()`, `vst_effects()`, `vst_arbpt()`, `vst_height()`, `vst_font()`, `vqt_f_extent()`, `vst_point()`

`v_ftext16()`

VOID `v_ftext16(handle, x, y, wstr, wstrlen)`

WORD `handle, x, y;`

WORD `*wstr;`

WORD `wstrlen;`

`v_ftext16()` is a variant binding of `v_ftext()` that outputs 16-bit Speedo character text rather than 8-bit ASCII text.

OPCODE 241

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the starting coordinate of the location to output text. *wstr* points to a **NULL**-terminated text string composed of **WORD**-sized Speedo characters. *wstrlen* specifies the length of the text string.

BINDING `WORD i;`

```
for( i = 0; i < wstrlen; i++)
    intin[i] = wstr[i];
```

```
contrl[0] = 241;
contrl[1] = 1;
contrl[3] = wstrlen;
contrl[6] = handle;
```

```
ptsin[0] = x;
ptsin[1] = y;
```

```
vdi();
```

COMMENTS This function should only be used when `vst_charmap()` has been used to indicate that **WORD**-sized Speedo character indexes should be recognized rather than 8-bit ASCII.

The text contained in *wstr* (including its **NULL** byte) should not exceed the maximum allowable size of the *intin* array (as indicated in the *work_out* array) or

the size of the *intin* array allocated by your compiler.

CAVEATS Current versions of **SpeedoGDOS** become confused when the space character (index 0) is encountered in the string. It is suggested that one of the three space characters (of varying widths) at indexes 560-562 be used instead.

SEE ALSO `v_ftext()`, `v_ftext_offset()`, `v_ftext_offset16()`, `v_gtext()`, `vst_alignment()`, `vst_color()`, `vst_effects()`, `vst_arbpt()`, `vst_height()`, `vst_font()`, `vqt_f_extent()`, `vst_point()`

v_ftext_offset()

VOID `v_ftext_offset(handle, x, y, str, offset)`

WORD `handle, x, y;`

char `*str;`

WORD `*offset;`

`v_ftext_offset()` is a variant binding of `v_ftext()` available under **SpeedoGDOS** which allows an offset vector for each character to be specified.

OPCODE 241

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* give the point where the string will be rendered. *offset* points to an array of **WORD**s which contains one *x* and *y* offset value for each character in *str*.

BINDING

```
WORD i = 0;

while(intin[i++] = (WORD)*str++;
--i;

ptsin[0] = x;
ptsin[1] = y;

for(j = 0; j < i * 2; j++)
    ptsin[j + 2] = offset[j];

contrl[0] = 241;
contrl[1] = i + 1;
contrl[3] = i;
contrl[6] = handle;

vdi();
```

COMMENTS The text contained in *str* (including its **NULL** byte) should not exceed the maximum allowable size of the *intin* array (as indicated in the *work_out* array) or

the size of the *intin* array allocated by your compiler.

To output 16-bit Speedo character indexes, use `v_ftext_offset16()`.

SEE ALSO `v_ftext_offset16()`, `v_ftext()`, `v_gtext()`

v_ftext_offset16()

VOID `v_ftext_offset(handle, x, y, wstr, wstrlen, offset)`

WORD *handle*, *x*, *y*;

WORD **wstr*;

WORD *wstrlen*;

WORD **offset*;

`v_ftext_offset16()` is a variant binding of `v_ftext_offset()` which allows 16-bit Speedo character strings to be output rather than 8-bit ASCII codes.

OPCODE 241

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* give the point where the string will be rendered. *offset* points to an array of **WORD**s which contains one *x* and *y* offset value for each character in *wstr*.

BINDING

```
WORD i;

for( i = 0; i < wstrlen; i++)
    intin[i] = wstr[i];

ptsin[0] = x;
ptsin[1] = y;

for(j = 0; j < i * 2; j++)
    ptsin[j + 2] = offset[j];

contrl[0] = 241;
contrl[1] = wstrlen + 1;
contrl[3] = wstrlen;
contrl[6] = handle;

vdi();
```

COMMENTS This function should only be used when `vst_charmap()` has been used to indicate that **WORD** sized Speedo character indexes should be recognized rather than 8-bit ASCII.

The text contained in *wstr* (including its **NULL** byte) should not exceed the

maximum allowable size of the *intin* array (as indicated in the *work_out* array) or the size of the *intin* array allocated by your compiler.

CAVEATS Current versions of **SpeedoGDOS** become confused when the space character (index 0) is encountered in the string. It is suggested that one of the three space characters (of varying widths) at indexes 560-562 be used instead.

SEE ALSO v_ftext16(), v_ftext_offset()

v_getbitmap_info()

VOID v_getbitmap_info(*handle*, *ch*, *advx*, *advy*, *xoff*, *yoff*, *width*, *height*, *bitmap*)

WORD *handle*, *ch*;

fix31 **advx*, **advy*, **xoff*, **yoff*;

WORD **width*, **height*;

VOID **bitmap*;

v_getbitmap_info() returns placement information for the bitmap of a character based on the current character font, size, and alignment.

OPCODE 239

AVAILABILITY Available only with **SpeedoGDOS**¹.

PARAMETERS *handle* specifies a valid workstation handle. *ch* is the character to return information about.

The **fix31** variables pointed to by *advx*, *advy*, *xoff*, and *yoff* will be filled in with the x and y advance and offset vectors respectively. The **WORD**s pointed to by *width* and *height* will be filled in with the width and height of the bitmap pointed to by the value returned in *bitmap*.

BINDING

```

contrl[0] = 239;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = ch;

vdi();

*width = intout[0];
*height = intout[1];
*advx = *(fix31 *)&intout[2];

```

¹This call did exist in **FSMGDOS**, however the call had a completely different calling format. Atari changed the existing call as no **FSMGDOS** program had yet been written to utilize it.

```
*advy = *(fix31 *)&intout[4];
*xoff = *(fix31 *)&intout[6];
*yoff = *(fix31 *)&intout[8];
*bitmap = *(void *)&intout[10];
```

COMMENTS The advance vector represents the amount to add to the current point to properly place the character. The offset vector, when added to the current point, give the location of the upper-left corner of the bitmap.

v_getoutline()

VOID v_getoutline(*handle*, *ch*, *xyarray*, *bezarray*, *maxverts*, *numverts*)

WORD *handle*, *ch*;

WORD **xyarray*;

char **bezarray*;

WORD *maxverts*;

WORD **numverts*;

v_getoutline() returns information about an **SpeedoGDOS** character required to generate the character with bezier curves.

OPCODE 243

AVAILABILITY Available only with **SpeedoGDOS**².

PARAMETERS *handle* specifies a valid workstation handle. *ch* specifies the character to return information about. The arrays pointed to by *xyarray* and *bezarray* are filled in with the bezier information for the character. The definition of *xyarray* and *bezarray* is given in the binding for v_bez().

maxverts should indicate the maximum number of vertices the buffer can hold. The **WORD** pointed to by *numverts* will be filled in with the actual number of vertices for the character.

BINDING

```
contrl[0] = 243;
contrl[1] = 0;
contrl[3] = 6;
contrl[6] = handle;

intin[0] = ch;
intin[1] = maxverts;
*(WORD *)&intin[2] = xyarray;
*(WORD *)&intin[4] = bezarray;

vdi();
```

²This call was present under **FSMGDOS**, however it's binding has dramatically changed. Applications using this binding will not operate under the older **FSMGDOS**.

```
*numverts = intout[0];
```

v_get_pixel()

VOID v_get_pixel(*handle*, *x*, *y*, *pindex*, *vindex*)

WORD *handle*, *x*, *y*;

WORD **pindex*, **vindex*;

v_get_pixel() returns the color value for a specified coordinate on the screen.

OPCODE 105

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. *x* any *y* specify the coordinate to return color information for.

In a palette-based mode the **WORD** pointed to by *pindex* will contain the hardware register index of the color and the **WORD** pointer to by *vindex* will contain the **VDI** index of the color.

In 16-bit true-color modes, *pindex* will be 0 and *vindex* will return the 16-bit RGB pixel value in the format {RRRR RGGG GGGB BBBB}.

In 32-bit color modes, the lower byte of *vindex* will contain the 8 bits of red data, the upper byte of *pindex* will contain the 8 bits of green data, and the lower byte of *pindex* will contain the 8 bits of blue data. The upper byte of *vindex* is reserved for non-color data.

BINDING

```
contrl[0] = 105;  
contrl[1] = 1;  
contrl[3] = 0;  
contrl[6] = handle;
```

```
ptsin[0] = x;  
ptsin[1] = y;
```

```
vdi();
```

```
*pindex = intout[0];  
*vindex = intout[1];
```

v_gtext()

VOID v_gtext(*handle*, *x*, *y*, *str*)

WORD *handle*, *x*, *y*;

char **str*;

v_gtext() outputs graphic text.

OPCODE 8

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the starting coordinates of the text (see **vst_alignment()**). *str* is a pointer to a **NULL**-terminated character string to print.

BINDING

```
WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 8;
contrl[1] = 1;
contrl[3] = --i;
contrl[6] = handle;

ptsin[0] = x;
ptsin[1] = y;

vdi();
```

COMMENTS The text contained in *str* (including its **NULL** byte) should not exceed the maximum allowable size of the *intin* array (as indicated in the *work_out* array) or the size of the *intin* array allocated by your compiler.

Using this function to output outline text with **FSMGDOS** is possible to remain backward-compatible but not recommended as it will introduce small errors as spacing remainders are lost.

SEE ALSO **v_ftext()**, **v_ftext_offset()**, **vst_color()**, **vst_effects()**, **vst_alignment()**, **vst_height()**, **vst_point()**

v_hardcopy()

VOID v_hardcopy(*handle*)

WORD *handle*;

v_hardcopy() invokes the ALT-HELP screen dump.

OPCODE 5

SUB-OPCODE 17

AVAILABILITY Supported by screen drivers running under ST compatible resolutions.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 17;
contrl[6] = handle;
```

vdi();

CAVEATS This function works in only ST compatible screen modes and should thus be avoided.

SEE ALSO Scrdmp()

v_hide_c()

VOID v_hide_c(*handle*)

WORD *handle*;

v_hide_c() hides the mouse cursor.

OPCODE 123

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 123;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;
```

vdi();

COMMENTS This call is nested. For each time you call this function you must call `v_show_c()` an equal number of times to show the mouse.

SEE ALSO `v_show_c()`, `graf_mouse()`

v_justified()

VOID `v_justified(handle, x, y, str, length, wflag, cflag)`

WORD `handle, x, y;`

char `*str;`

WORD `length, wflag, cflag;`

`v_justified()` outputs justified graphics text.

OPCODE 11

SUB-OPCODE 10

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by `v_opnvwk()` or `v_opnwk()`.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the starting coordinates at which to draw the **NULL**-terminated text string (see `vst_alignment()`) pointed to by *str*. *length* specifies the pixel length of the area to justify on.

wflag and *cflag* specify the type of justification to perform between words and characters respectively. A value of **NOJUSTIFY** (0) indicates no justification whereas a value of **JUSTIFY** (1) indicates to perform justification.

BINDING

```
WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 11;
contrl[1] = 2;
contrl[3] = --i;
contrl[5] = 10;
contrl[6] = handle;

intin[0] = wflag;
intin[1] = cflag;

ptsin[0] = x;
```

```
ptsin[1] = y;  
ptsin[2] = length;  
ptsin[3] = 0;  
  
vdi();
```

COMMENTS This call does not take into account remainder information from outline fonts.

SEE ALSO v_gtext(), v_ftext(), vst_color(), vst_font(), vst_effects(), vst_alignment(), vst_point(), vst_height()

v_killoutline()

VOID v_killoutline(*handle*, *outline*)

WORD *handle*;

FSMOUTLINE *outline*;

v_killoutline() releases an outline from memory.

OPCODE 242

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

COMMENTS Under **FSMGDOS** this call was required to release memory allocated for an outline returned from v_getoutline(). With **SpeedoGDOS**, this call is no longer required and is thus not documented further.

SEE ALSO v_getoutline()

v_loadcache()

WORD v_loadcache(*handle*, *fname*, *mode*)

WORD *handle*;

char **fname*;

WORD *mode*;

v_loadcache() loads a previously saved cache file from disk.

OPCODE 250

AVAILABILITY Supported only by **FSMGDOS** and **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *fname* specifies the **GEMDOS** file

specification of the cache file to load. *mode* specifies whether current data will be flushed first. A value of 0 will append the loaded cache to the current cache whereas a value of 1 will flush the cache prior to loading.

BINDING

```
WORD i = 1;

intin[0] = mode;
while(intin[i++] = (WORD)*fname++);

contrl[0] = 250;
contrl[1] = 0;
contrl[3] = --i;
contrl[6] = handle;

vdi();

return intout[0];
```

RETURN VALUE `v_loadcache()` returns 0 if successful or -1 if an error occurred.

COMMENTS This command only affects the cache responsible for storing bitmaps created from outline characters.

SEE ALSO `v_savecache()`, `v_flushcache()`

v__meta_extents()

VOID `v__meta_extents(handle, xmin, ymin, xmax, ymax)`

WORD `handle, xmin, ymin, xmax, ymax;`

`v__meta_extents()` embeds placement information for a metafile.

OPCODE 5

SUB-OPCODE 98

AVAILABILITY Supported by all metafile drivers.

PARAMETERS *handle* specifies a valid workstation handle. *xmin* and *ymin* specify the upper left corner of the bounding box of the metafile. *xmax* and *ymax* specify the lower left corner.

BINDING

```
contrl[0] = 5;
contrl[1] = 2;
contrl[3] = 0;
contrl[5] = 98;
contrl[6] = handle;

ptsin[0] = xmin;
```

```

ptsin[1] = ymin;
ptsin[2] = xmax;
ptsin[3] = ymax;

vdi();

```

COMMENTS Parameters sent to this call should be specified in whatever coordinate system the metafile is currently using.

SEE ALSO `vm_pagesize()`

v_opnvwk()

VOID v_opnvwk(*work_in*, *handle*, *work_out*)

WORD **work_in*, **handle*, **work_out*;

v_opnvwk() opens a virtual **VDI** workstation.

OPCODE 100

AVAILABILITY Supported by all drivers.

PARAMETERS *work_in* is a pointer to an array of 11 **WORD**s which define the initial defaults for the workstation as follows:

<i>work_in[x]</i>	Meaning
0	Device identification number. This indicates the physical device ID of the device (the line number of the driver in ASSIGN.SYS when using GDOS). For screen devices you should normally use the value Getrez() + 2, however, a value of 1 is acceptable if not using any loaded fonts.
1	Default line type (same as vsl_type()).
2	Default line color (same as vsl_color()).
3	Default marker type (same as vsm_type()).
4	Default marker color (same as vsm_color()).
5	Default font (same as vst_font()).
6	Default text color (same as vst_color()).
7	Default fill interior.
8	Default fill style.
9	Default fill color.

10	Coordinate type flag. A value of 0 specifies NDC 'Normalized Device Coordinates' coordinates whereas a value of 2 specifies RC 'Raster Coordinates'. All other values are reserved. NDC coordinates are only available when using external drivers with GDOS .
----	---

handle should be set to the current handle (not the device ID) of the physical workstation for this device. For screen devices this is the value returned by **graf_handle()**. On exit *handle* will be filled in the **VDI** workstation handle allocated, if successful, or 0 if the workstation could not be opened.

work_out points to an array of 57 **WORD**s which on exit will be filled in by the **VDI** with information regarding the allocated workstation as follows (a structure name is listed beside its array member for those using the 'C' style **VDI_Workstation** structure instead of the array):

VDI Structure		
<i>work_out[x]</i>	Member	Meaning
0	<i>xres</i>	Width of device in pixels - 1.
1	<i>yres</i>	Height of device in pixels - 1.
2	<i>noscale</i>	Device coordinate units flag: 0 = Device capable of producing a precisely scaled image (screen, printer, etc...) 1 = Device not capable of producing a precisely scaled image (film recorder, etc...)
3	<i>wpixel</i>	Width of pixel in microns (1/25400 inch).
4	<i>hpixel</i>	Height of pixel in microns (1/25400 inch).
5	<i>cheights</i>	Number of character heights (0 = continuous scaling).
6	<i>linetypes</i>	Number of line types.
7	<i>linewidths</i>	Number of line widths (0 = continuous scaling).
8	<i>markertypes</i>	Number of marker types.
9	<i>markersizes</i>	Number of marker sizes (0 = continuous scaling).
10	<i>faces</i>	Number of faces supported by the device.
11	<i>patterns</i>	Number of available patterns.
12	<i>hatches</i>	Number of available hatches.
13	<i>colors</i>	Number of predefined colors/pens (ST High = 2, ST Medium = 4, TT Low = 256, True Color = 256).
14	<i>ngdps</i>	Number of supported GDP's

15-24	<i>cangdps[10]</i>	<i>cangdps</i> [0 – (<i>ngdps</i> - 1)] contains a list of the GDP's the device supports as follows: 1 = Bar 2 = Arc 3 = Pie Slice 4 = Circle 5 = Ellipse 6 = Elliptical Arc 7 = Elliptical Pie 8 = Rounded Rectangle 9 = Filled Rounded Rectangle 10 = Justified Graphics Text
25-34	<i>gdpatr[10]</i>	For each GDP as listed above, <i>gdpatr</i> [0 – (<i>ngdps</i> - 1)] indicates the attributes which are applied to that GDP as follows: 1 = Polyline (<i>vsl_...</i>) 2 = Polymarker (<i>vsm_...</i>) 3 = Text (<i>vst_...</i>) 4 = Fill Area (<i>vsf_...</i>) 5 = None
35	<i>cancolor</i>	Color capability flag. 0 = No 1 = Yes
36	<i>cantextrot</i>	Text rotation flag. 0 = No 1 = Yes
37	<i>canfillarea</i>	Fill area capability flag. 0 = No 1 = Yes
38	<i>cancellarray</i>	Cell array capability flag. 0 = No 1 = Yes
39	<i>palette</i>	Number of available colors in palette. 0 = > 32767 colors 2 = Monochrome >2 = Color
40	<i>locators</i>	Number of locator devices. 1 = Keyboard only. 2 = Keyboard and other.
41	<i>valuators</i>	Number of valuator devices. 1 = Keyboard only. 2 = Keyboard and other.
42	<i>choicedevs</i>	Number of choice devices. 1 = Function keys. 2 = Function keys + keypad.
43	<i>stringdevs</i>	Number of string devices. 1 = Keyboard.
44	<i>wstype</i>	Workstation type. 0 = Output only 1 = Input only 2 = Input/Output 3 = Metafile
45	<i>minwchar</i>	Minimum character width in pixels.
46	<i>minhchar</i>	Minimum character height in pixels.
47	<i>maxwchar</i>	Maximum character width in pixels.

7.64 – VDI/GDOS Function Reference

48	<i>maxhchar</i>	Maximum character height in pixels.
49	<i>minwline</i>	Minimum line width.
50	<i>zero5</i>	Reserved (0).
51	<i>maxwline</i>	Maximum line width.
52	<i>zero7</i>	Reserved (0).
53	<i>minwmark</i>	Minimum marker width.
54	<i>minhmark</i>	Minimum marker height.
55	<i>maxwmark</i>	Maximum marker width.
56	<i>maxhmark</i>	Maximum marker height.

BINDING

```
WORD i;  
  
contrl[0] = 100;  
contrl[1] = 0;  
contrl[3] = 11;  
contrl[6] = *handle;  
  
for(i = 0; i < 11; i++)  
    intin[i] = work_in[i];  
  
vdi();  
  
*handle = contrl[6];  
  
for(i = 0; i < 45; i++)  
    work_out[i] = intout[i];  
  
for(i = 0; i < 13; i++)  
    work_out[45+i] = intout[i];
```

CAVEATS

The **VDI** included with **TOS** versions less than 2.06 sometimes returned the same handle for consecutive calls using the same physical handle.

COMMENTS

Using multiple virtual workstations provides the benefit of being able to define multiple sets of default line types, text faces, etc... without having to constantly set them.

The **VDI_Workstation** structure method is the recommended method of using this function. See the **VDI** entry for **V_Opnwkw()** and **V_Opnvwkw()**.

Desk accessories running under **TOS** versions below 1.4 should not leave a workstation open across any call which might surrender control to **GEM** (**evnt_button()**, **evnt_multi()**, etc...). This could give **GEM** time to change screen resolutions and **TOS** versions below 1.4 did not release memory allocated by a desk accessory (including workstations) when a resolution change occurred.

SEE ALSO

v_opnwkw(), **vq_extend()**, **v_clsvwkw()**, **V_Opnvwkw()**

V_Opnmwk()

WORD V_Opnmwk(*dev*)

VDI_Workstation *dev*;

V_Opnmwk() is not a component of the **VDI**, rather an interface binding designed to simplify working with virtual screen workstations. It will open a virtual screen workstation with a **VDI_Workstation** structure as a parameter rather than *work_in* and *work_out* arrays.

OPCODE N/A

AVAILABILITY User-defined.

PARAMETERS *ws* is a pointer to a **VDI_Workstation** structure defined as follows (for the meaning of each structure member, refer to **v_opnmwk()**):

```
typedef struct
{
    WORD handle, dev_id;
    WORD wchar, hchar, wbox, hbox;
    WORD xres, yres;
    WORD noscale;
    WORD wpixel, hpixel;
    WORD cheights;
    WORD linetypes, linewidths;
    WORD markertypes, markersizes;
    WORD faces, patterns, hatches, colors;
    WORD ngdps;
    WORD cangdps[10];
    WORD gdpatrr[10];
    WORD cancelor, cantextrot;
    WORD canfillarea, cancellarray;
    WORD palette;
    WORD locators, valuator;
    WORD choicedevs, stringdevs;
    WORD wstype;
    WORD minwchar, minhchar;
    WORD maxwchar, maxwchar;
    WORD minwline;
    WORD zero5;
    WORD maxwline;
    WORD zero7;
    WORD minwmark, minhmark;
    WORD maxwmark, maxhmark;
    WORD screentype;
    WORD bgcolors, textfx;
    WORD canscale;
    WORD planes, lut;
    WORD rops;
    WORD cancontourfill, textrot;
    WORD writemodes;
    WORD inputmodes;
}
```

```
        WORD textalign, inking, rubberbanding;
        WORD maxvertices, maxintin;
        WORD mousebuttons;
        WORD widestyles, widemodes;
        WORD reserved[38];
    } VDI_Workstation;
```

BINDING

```
WORD
V_Opnvwk( dev )
VDI_Workstation dev;
{
    WORD i, in[11];

    in[0] = Getrez() + 2;
    dev->dev_id = in[0];
    for(i = 1; i < 10; in[i++] = 1);
    in[10] = 2;
    i = graf_handle( &dev->wchar,
                    &dev->hchar, &dev->wbox,
                    &dev->hbox );

    v_opnvwk( in, &i, &dev->xres );
    dev->handle = i;

    if(i)
        vq_extnd( i, 1, &dev->screentype );

    return (i);
}
```

RETURN VALUE **V_Opnvwk()** returns 0 if non-successful or the workstation handle otherwise.

COMMENTS This function definition is adapted from an article which appeared in the ‘Atari RSC’ developers newsletter (Nov ‘90 - Jan ‘91).

SEE ALSO **v_opnvwk()**, **V_Opnwk()**, **vq_extnd()**

v_opnvwk()

VOID v_opnvwk(*work_in*, *handle*, *work_out*)

WORD **work_in*, **handle*, **work_out*;

v_opnvwk() opens a physical workstation.

OPCODE 1

AVAILABILITY Available only with some form of **GDOS**.

PARAMETERS All parameters for this function are consistent with **v_opnvwk()** except as follows:

On entry, *handle* does not need to contain any specific value. On return, however,

it will contain a workstation handle if successful or 0 if the call failed.

BINDING

```
WORD i;

contrl[0] = 1;
contrl[1] = 0;
contrl[3] = 11;

for(i = 0; i < 11; i++)
    intin[i] = work_in[i];

vdi();

*handle = contrl[6];

for(i = 0; i < 45; i++)
    work_out[i] = intout[i];

for(i = 0; i < 13; i++)
    work_out[45+i] = ptsout[i];
```

COMMENTS

Physical workstations should be opened when needed and closed immediately afterwards. For example, a word processor should *not* open the printer workstation when the application starts and close it when it ends. If this is done, the user will be unable to change printers with the Printer Setup CPX(s).

SEE ALSO

V_Opnwk(), v_opnvwk(), vq_extnd()

V_Opnwk()

WORD V_Opnwk(*devno*, *dev*)

WORD *devno*;

VDI_Workstation *dev*;

V_Opnwk() is not a component of the **VDI**, rather an interface binding designed to simplify working with **VDI** workstations. It will open a physical workstation using a **VDI_Workstation** structure rather than *work_in* and *work_out*.

OPCODE

N/A

AVAILABILITY

User-defined.

PARAMETERS

devno specifies the device ID of the device to open. Valid values for *devno* follow:

1-10	=	Screen (loaded device drivers only)
11-20	=	Plotters
21-30	=	Printers
31-40	=	Metafile Drivers

41-50 = Camera Drivers
51-60 = Tablet Drivers
61-70 = Memory Drivers

ws is a **VDI_Workstation** structure as defined in **V_Opnvwk()**.

BINDING

```
WORD
V_Opnvwk( devno, dev )
WORD devno;
VDI_Workstation dev;
{
    WORD i, in[11];

    in[0] = dev->dev_id = devno;
    for(i = 1; i < 10; in[i++] = 1);
    in[10] = 2;
    i = devno;

    v_opnvwk( in, &i, &dev->xres );
    dev->handle = i;

    if(i)
        vq_extnd( i, 1, &dev->screentype );

    return (i);
}
```

RETURN VALUE **V_Opnvwk()** returns a workstation handle if successful or 0 if the call failed.

COMMENTS This function definition is adapted from an article which appeared in the ‘Atari .RSC’ developers newsletter (Nov ‘90 - Jan ‘91).

SEE ALSO **v_opnvwk()**, **vq_extnd()**, **v_opnvwk()**, **V_Opnvwk()**

v_output_window()

VOID **v_output_window(handle, pxy)**

WORD *handle*;

WORD **pxy*;

v_output_window() outputs a specified portion of the current page.

OPCODE 5

SUB-OPCODE 22

AVAILABILITY Supported by all printer and metafile drivers under any type of **GDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *pxy* is a pointer to an array of four

WORDS in **VDI** rectangle format which specifies the bounding extents of the current page to output.

BINDING

```

contrl[0] = 5;
contrl[1] = 2;
contrl[3] = 0;
contrl[5] = 21;
contrl[6] = handle;

```

```

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];

```

```
vdi();
```

CAVEATS Some printer drivers ignore the sides of the bounding box specified and print the entire width of the page.

COMMENTS This call is similar to **v_updwk()** except that only a portion of the page is output.

SEE ALSO **v_updwk()**

v_pgcount()

VOID v_pgcount(*handle*, *numcopies*)

WORD *handle*, *numcopies*;

v_pgcount() is used to cause the laser printer to output multiple copies of the current page.

OPCODE 5

SUB-OPCODE 2000

AVAILABILITY Supported only with some laser printer drivers (for instance the Atari laser printer driver) under some form of **GDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *numcopies* specifies the number of copies to print minus one. A value of 0 means print one copy, a value of 1, two copies, and so on.

BINDING

```

contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 1;
contrl[5] = 2000;
contrl[6] = handle;

intin[0] = numcopies;

```

```
vdi();
```

COMMENTS This call is preferred over repeatedly calling `v_updwk()` and `v_form_adv()` as this method forces the printer data to be resent for each page.

v_pieslice()

VOID `v_pieslice(handle, x, y, radius, startangle, endangle)`

WORD `handle, x, y, radius, startangle, endangle;`

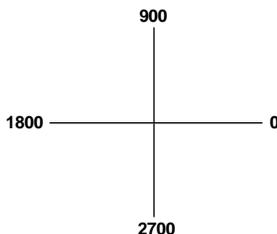
`v_pieslice()` outputs a filled pie segment.

OPCODE 11

SUB-OPCODE 3

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by `v_opnvwk()` or `v_opnwk()`.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* specify the center of a circular segment of radius *radius* which is drawn between the angles of *startangle* and *endangle* (specified in tenths of degrees - legal values illustrated below) and connected to the center point.



BINDING

```
contrl[0] = 11;  
contrl[1] = 4;  
contrl[3] = 2;  
contrl[5] = 3;  
contrl[6] = handle;
```

```
ptsin[0] = x;  
ptsin[1] = y;  
ptsin[2] = ptsin[3] = ptsin[4] = ptsin[5] = 0  
ptsin[6] = radius;
```

```
intin[0] = startangle;
```

```
intin[1] = endangle;  
vdi();
```

SEE ALSO **v_ellipse(), vsf_color(), vsf_style(), vsf_interior(), vsf_udpat(), vsf_perimeter()**

v_pline()

VOID v_pline(*handle*, *count*, *pxy*)

WORD *handle*, *count*;

WORD **pxy*;

v_pline() outputs a polyline (group of one or more lines).

OPCODE 6

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *count* specifies the number of vertices in the line path (2 to plot a single line). *pxy* points to a **WORD** array with *count* * 2 elements containing the vertices to plot as in (X1, Y1), (X2, Y2), etc...

BINDING

```
WORD i;  
  
contrl[0] = 6;  
contrl[1] = count;  
contrl[3] = 0;  
contrl[6] = handle;  
  
for(i = 0; i < (count*2); i++)  
    ptsin[i] = count[i];  
  
vdi();
```

COMMENTS To draw a single point with this function, *pxy[2]* should equal *pxy[0]*, *pxy[3]* should equal *pxy[1]*, and *count* should be 2.

SEE ALSO **v_fillarea(), vsl_color(), vsl_type(), vsl_udsty(), vsl_ends()**

v_pmarker()

VOID v_pmarker(*handle*, *count*, *pxy*)

WORD *handle*, *count*;

WORD **pxy*;

v_pmarker() outputs one or several markers.

OPCODE 7

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation. *count* specifies the number of markers to plot. *pxy* points to a **WORD** array with (*count* * 2) elements containing the vertices of the markers to plot as in (X1, Y1), (X2, Y2), etc...

BINDING

```
WORD i;  
  
contrl[0] = 7;  
contrl[1] = count;  
contrl[3] = 0;  
contrl[6] = handle;  
  
for(i = 0; i < (count * 2); i++)  
    ptsin[i] = pxy[i];  
  
vdi();
```

COMMENTS Single points may be plotted quickly with this function when the proper marker type is selected with **vsm_type()**.

SEE ALSO **vsm_type()**, **vsm_height()**, **vsm_color()**

v_rbox()

VOID v_rbox(*handle*, *pxy*)

WORD *handle*;

WORD **pxy*;

v_rbox() outputs a rounded box (not filled).

OPCODE 11

SUB-OPCODE 8

AVAILABILITY	Supported by all drivers. This function composes one of the 10 VDI GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by v_opnvwk() or v_opnwk() .
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>pxy</i> points to an array of 4 WORDS containing the VDI format rectangle of the rounded box to output.
BINDING	<pre> contrl[0] = 11; contrl[1] = 2; contrl[3] = 0; contrl[5] = 8; contrl[6] = handle; ptsin[0] = pxy[0]; ptsin[1] = pxy[1]; ptsin[2] = pxy[2]; ptsin[3] = pxy[3]; vdi(); </pre>
CAVEATS	There is no way to define to size of the 'roundness' of the corners.
SEE ALSO	v_rfbox() , v_bar() , vsl_type() , vsl_color() , vsl_udsty() , vsl_ends()

v_rfbox()

VOID v_rfbox(*handle*, *pxy*)

WORD *handle*;

WORD **pxy*;

v_rfbox() outputs a filled rounded-rectangle.

OPCODE 11

SUB-OPCODE 9

AVAILABILITY Supported by all drivers. This function composes one of the 10 **VDI** GDP's (Generalized Drawing Primitives). Although all current drivers support all GDP's, their availability is not guaranteed and may vary. To check for a particular GDP refer to the table returned by **v_opnvwk()** or **v_opnwk()**.

PARAMETERS *handle* specifies a valid workstation handle. *pxy* points to an array of four **WORDS** which specify the **VDI** format rectangle of the rounded-rectangle to output.

BINDING

```
contrl[0] = 11;
```

```
contrl[1] = 2;  
contrl[3] = 0;  
contrl[5] = 9;  
contrl[6] = handle;  
  
ptsin[0] = pxy[0];  
ptsin[1] = pxy[1];  
ptsin[2] = pxy[2];  
ptsin[3] = pxy[3];  
  
vdi();
```

CAVEATS There is no way to specify the ‘roundness’ of the rectangle.

SEE ALSO [v_rbox\(\)](#), [v_bar\(\)](#), [vsf_color\(\)](#), [vsf_style\(\)](#), [vsf_interior\(\)](#), [vsf_udpat\(\)](#)

v_rmcur()

VOID [v_rmcur\(\)](#) (*handle*)

WORD *handle*;

[v_rmcur\(\)](#) removes the last mouse cursor displayed.

OPCODE 5

SUB-OPCODE 19

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 19;  
contrl[6] = handle;  
  
vdi();
```

COMMENTS [v_rmcur\(\)](#) should only be used in conjunction with [v_dspcur\(\)](#) when the mouse is moved manually. [graf_mouse\(\)](#) or [v_hide_c\(\)](#) should be used unless this is your intention.

SEE ALSO [v_hide_c\(\)](#), [graf_mouse\(\)](#)

v_rvoff()

VOID v_rvoff(*handle*)
WORD *handle*;

v_rvoff() causes alpha screen text to be displayed in normal video (as opposed to inverse).

OPCODE 5

SUB-OPCODE 14

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 14;  
contrl[6] = handle;  
  
vdi();
```

COMMENTS This call is equivalent to the ESC-Q VT-52 code.

SEE ALSO v_rvon(), v_curtext()

v_rvon()

VOID v_rvon(*handle*)
WORD *handle*;

v_rvon() causes alpha screen text to be displayed in inverse mode.

OPCODE 5

SUB-OPCODE 13

AVAILABILITY Supported by all screen devices.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;
```

```
    contrl[5] = 13;
    contrl[6] = handle;

    vdi();
```

COMMENTS This call is equivalent to the ESC-P VT-52 code.

SEE ALSO `v_rvoff()`, `v_curtext()`

v_savecache()

WORD `v_savecache(handle, fname)`

WORD `handle`;

char `*fname`;

`v_savecache()` saves the current outline cache.

OPCODE 249

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS `handle` specifies a valid workstation handle. `fname` specifies the **GEMDOS** file specification of the cache file to save.

BINDING

```
WORD i = 0;

while(intin[i++] = (WORD)*fname++);

    contrl[0] = 249;
    contrl[1] = 0;
    contrl[3] = --i;
    contrl[6] = handle;

    vdi();

    return intout[0];
```

RETURN VALUE `v_savecache()` returns 0 if successful or -1 if an error occurred.

COMMENTS This call only saves the portion of the cache responsible for storing bitmaps created from outlines.

SEE ALSO `v_loadcache()`, `v_flushcache()`

v_set_app_buff()

VOID v_set_app_buff(*but*, *nparagraphs*)

VOID **buf*;

WORD *nparagraphs*;

`v_set_app_buff()` designates memory for use by the bezier generation routines.

OPCODE -1

SUB-OPCODE 6

AVAILABILITY Available only with **FONTGDOS**, **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *buf* specifies the address of a buffer which the bezier generator routines may safely use. *nparagraphs* specifies the size of the buffer in ‘paragraphs’ (16 bytes).

BINDING

```
contrl[0] = -1;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 6;

*(VOID *)&intin[0] = buf;
intin[2] = nparagraphs;

vdi();
```

COMMENTS Before the application exits, it should call `v_set_app_buff(NULL, 0)` to ‘unmark’ memory. The application is then responsible for deallocating the memory.

In the absence of this call the first `v_bez()` or `v_bezfill()` call will allocate its own buffer of 8K. Atari documentation recommends a size of about 9K depending on the extents of the bezier you wish to generate.

SEE ALSO `v_bez()`

v_show_c()

VOID v_show_c(*handle*, *reset*)

WORD *handle*, *reset*;

`v_show_c()` ‘unhides’ the mouse cursor.

OPCODE 122

AVAILABILITY	Supported by all screen drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. If <i>reset</i> is 0 the mouse will be displayed regardless of the number of times it was 'hidden'. Otherwise, the call will only display the cursor if the function has been called an equal number of times compared to v_hide_c() .
BINDING	<pre>contrl[0] = 122; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; intin[0] = reset; vdi();</pre>
CAVEATS	While it may be tempting to always use a <i>reset</i> value of 0, it is not recommended. Doing so may confuse the system so that when the critical error handler is called, the mouse is not displayed.
SEE ALSO	v_hide_c() , graf_mouse()

v_updwk()

VOID v_updwk(*handle*)

WORD *handle*;

v_updwk() outputs the current page to the specified device.

OPCODE	4
AVAILABILITY	Supported by all printer, metafile, plotter, and camera devices when using any form of GDOS .
PARAMETERS	<i>handle</i> specifies a valid workstation handle.
BINDING	<pre>contrl[0] = 4; contrl[1] = contrl[3] = 0; contrl[6] = handle; vdi();</pre>
COMMENTS	This call does not cause the 'page' to be ejected. You must use either v_clrwk() or v_form_adv() to accomplish that.
SEE ALSO	v_clrwk() , v_form_adv()

v_write_meta()

VOID v_write_meta(*handle*, *intin_len*, *intin*, *ptsin_len*, *ptsin*)

WORD *handle*, *intin_len*;

WORD **intin*;

WORD *ptsin_len*;

WORD **ptsin*;

v_write_meta() writes a customized metafile sub-opcode.

OPCODE 5

SUB-OPCODE 99

AVAILABILITY Supported by all metafile drivers.

PARAMETERS *handle* specifies a valid workstation handle. *intin* points to an array of **WORDS** with *intin_len* (0-127) elements. *ptsin* points to an array of **WORDS** with *ptsin_len* (0-127) elements. *ptsin* is not required to be of any length, however, *intin* should be at least one word long to specify the sub-opcode in *intin[0]*. Sub-opcodes 0-100 are reserved for use by Atari. Several pre-defined sub-opcodes in this range already exist as follows:

Sub-Opcode:	
intin[0]	Meaning
10	Start group.
11	End group.
49	Set no line style.
50	Set attribute shadow on.
51	Set attribute shadow off.
80	Start draw area type primitive.
81	End draw area type primitive.

BINDING

```
WORD i;

contrl[0] = 5;
contrl[1] = ptsin_len;
contrl[3] = intin_len;
contrl[5] = 99;
contrl[6] = handle;

for(i = 0; i < intin_len; i++)
    intin[i] = m_intin[i];
for(i = 0; i < ptsin_len; i++)
    ptsin[i] = m_ptsin[i];
```

```
vdi();
```

COMMENTS Metafile readers should ignore and safely skip any opcodes not understood.

vex_butv()

VOID vex_butv(*handle*, *butv*, *old_butv*)

WORD *handle*;

WORD (**butv*)(**WORD** *bstate*);

WORD (***old_butv*)(**WORD** *bstate*);

vex_butv() installs a routine which is called by the **VDI** every time a mouse button is pressed.

OPCODE 125

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid physical workstation handle. *butv* points to a user-defined button-click handler routine. The address pointed to by *old_butv* will be filled in with the address of the old button-click handler.

BINDING

```
contrl[0] = 125;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)butv >> 16);
contrl[8] = (WORD)((LONG)butv);

vdi();

*(LONG *)old_butv = (LONG)(((LONG)contrl[9] << 16) |
    (LONG)contrl[10]);
```

COMMENTS Upon entry to *butv*, the mouse status is contained in 68x00 register D0 (in the same format as the button return value in **vq_mouse()**). A ‘C’ handler should, therefore, be sure to specify register calling parameters for this function. Any registers which will be modified should be saved and restored upon function exit. The routine may call the **BIOS** and/or **XBIOS** sparingly but should not call the **AES**, **VDI**, or **GEMDOS**.

SEE ALSO **vex_curv()**, **vex_motv()**

vex_curv()

```
VOID vex_curv( handle, curv, old_curv )  
WORD handle;  
WORD (*curv)( (WORD) mx, (WORD) my );  
WORD (**old_curv)( (WORD) mx, (WORD) my );
```

vex_curv() installs a routine which is called every time the mouse cursor is drawn allowing a customized mouse rendering routine to replace that of the system.

OPCODE 126

AVAILABILITY Supported by all screen devices.

PARAMETERS *handle* specifies a valid physical workstation handle. *curv* points to a user defined function which will be called every time the mouse is to be refreshed. *old_curv* is the address of a pointer to the old rendering routine which will be filled in by the function on exit.

BINDING

```
contrl[0] = 126;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;  
contrl[7] = (WORD)((LONG)curv >> 16);  
contrl[8] = (WORD)((LONG)curv);  
  
vdi();  
  
*(LONG *)old_curv = (LONG)(((LONG)contrl[9] << 16) |  
                        (LONG)contrl[10]);
```

COMMENTS Upon entry to *curv*, the mouse's X and Y location on screen is contained in 68x00 registers D0 and D1 respectively. A 'C' handler should, therefore, be sure to specify register calling parameters for this function. Any registers which will be modified should be saved and restored upon function exit. The routine may call the **BIOS** and/or **XBIOS** sparingly but should not call the **AES**, **VDI**, or **GEMDOS**.

SEE ALSO **vex_butv()**, **vex_motv()**

vex_motv()

VOID vex_motv(*handle*, *motv*, *old_motv*)
WORD *handle*;
WORD (**motv*)((WORD) *mx*, (WORD) *my*);
WORD (***old_motv*)((WORD) *mx*, (WORD) *my*);

vex_motv() installs a user routine which is called every time the mouse pointer is moved.

OPCODE 126

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid physical workstation handle. *motv* points to a user-defined routine which is called every time the mouse is moved. *old_motv* is an address to a pointer which will be filled in containing the address of the old function.

BINDING

```
contrl[0] = 126;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;  
contrl[7] = (WORD)((LONG)motv >> 16);  
contrl[8] = (WORD)((LONG)motv);  
  
vdi();  
  
*(LONG *)old_motv = (LONG)(((LONG)contrl[9] << 16) |  
    (LONG)contrl[10]);
```

COMMENTS Upon entry to *motv*, the mouse's new X and Y location is contained in 68x00 registers D0 and D1 respectively. A 'C' handler should, therefore, be sure to specify register calling parameters for this function. Any registers which will be modified should be saved and restored upon function exit. The routine may call the **BIOS** and/or **XBIOS** sparingly but should not call the **AES**, **VDI**, or **GEMDOS**. The routine may modify the contents of D0 and D1 as necessary to affect the movement of the mouse (one way of implementing a mouse accelerator).

SEE ALSO **vex_curv()**, **vex_butv()**

vex_timv()

```
VOID vex_timv( handle, timv, old_timv, mpt )
WORD handle;
VOID (*timv)( VOID );
VOID (**old_timv)( VOID );
WORD *mpt;
```

vex_timv() installs a user-defined routine that will be called at each timer tick (currently once every 50 milliseconds).

OPCODE 118

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid physical workstation handle. *timv* should point to a user-defined timer tick routine. *old_timv* is an address to a pointer which will be filled in with the old timer tick routine. *mpt* is a pointer to a **WORD** which will be filled in with the value representing the current number of milliseconds per timer tick.

BINDING

```
contrl[0] = 118;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)timv >> 16);
contrl[8] = (WORD)((LONG)timv);

vdi();

*(LONG *)old_timv = (LONG)(((LONG)contrl[9] << 16) |
    (LONG)contrl[10]);
```

COMMENTS Any registers which will be modified should be saved and restored upon function exit. The routine may call the **BIOS** and/or **XBIOS** sparingly but should not call the **AES**, **VDI**, or **GEMDOS**. The routine should fall through to the old routine. As this vector is jumped through quite often, the routine should be very simple to avoid system performance slowdowns.

vm_coords()

```
VOID vm_coords( handle, xmin, ymin, xmax, ymax )
WORD handle, xmin, ymin, xmax, ymax;
```

vm_coords() allows the use of variable coordinate systems with metafiles.

OPCODE 5

SUB-OPCODES	99, 1
AVAILABILITY	Supported by all metafile drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>xmin</i> and <i>ymin</i> specify the coordinate pair which provides an anchor for the upper-left point of the coordinate system. <i>xmax</i> and <i>ymax</i> specify the coordinate pair which provides an anchor for the lower-right point of the coordinate system.
BINDING	<pre>contrl[0] = 5; contrl[1] = 0; contrl[3] = 5; contrl[5] = 99; contrl[6] = handle; intin[0] = 1; intin[1] = xmin; intin[2] = ymin; intin[3] = xmax; intin[4] = ymax; vdi();</pre>
COMMENTS	Use of this function allows the use of practically any coordinate system with a limit of (-32768, -32768), (32767, 32767). Metafiles default to a coordinate space of (0, 32767), (32767, 0).
SEE ALSO	vm_pagesize() , v_meta_extents()

vm_filename()

VOID vm_filename(*handle*, *fname*)

WORD *handle*;

char **fname*;

vm_filename() allows specifying a user-defined filename for metafile output.

OPCODE 5

SUB-OPCODE 100

AVAILABILITY Supported by all metafile drivers.

PARAMETERS *handle* specifies a valid workstation handle. *fname* points to a **NULL**-terminated **GEMDOS** filename which all metafile output should be redirected to.

BINDING

```
WORD i = 0;

while(intin[i++] = (WORD)*fname++);

contrl[0] = 5;
contrl[1] = 0;
contrl[3] = --i;
contrl[5] = 100;
contrl[6] = handle;

vdi();
```

CAVEATS When a metafile is opened, the default file ‘GEMFILE.GEM’ is created in the current **GEMDOS** path on the current drive and is not deleted as a result of this call. You will need to manually delete it yourself.

COMMENTS This call should be made immediately after a **v_opnwk()** to a metafile handle if you wish to use an alternate filename to prevent data from being lost.

vm_pagesize()

VOID vm_pagesize(*handle*, *pwidth*, *pheight*)

WORD *handle*, *pwidth*, *pheight*;

vm_pagesize() specifies a metafile’s source page size.

OPCODE 5

SUB-OPCODES 99, 0

AVAILABILITY Supported by all metafile drivers.

PARAMETERS *handle* specifies a valid workstation handle. *pwidth* specifies the width of the page which the metafile was originally placed on in tenths of a millimeter. *pheight* specifies the height of the page which the metafile was originally placed on in tenths of a millimeter.

BINDING

```
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 99;
contrl[6] = handle;

intin[0] = 0;
intin[1] = pwidth;
intin[2] = pheight;

vdi();
```

COMMENTS A metafile originally designed on an 8.5” x 11” page would have a *pwidth* value of 2159 and a *pheight* value of 2794.

SEE ALSO `v_meta_extents()`

vq_cellarray()

VOID `vq_cellarray(handle, pxy, rowlen, num_rows, elements, rows_used, status, colarray)`

WORD *handle*;

WORD **pxy*;

WORD *rowlen, num_rows*;

WORD **elements, *rows_used, *status, *colarray*;

`vq_cellarray()` returns the cell array definitions of specified pixels.

OPCODE 27

AVAILABILITY Not supported by any known drivers.

PARAMETERS *handle* specifies a valid workstation handle. *pxy* points to an array of 4 **WORD**s which specify a **VDI** format rectangle. *row_length* specifies the length of each row in the color array. *num_rows* specifies the number of total rows in the color array.

Upon return, the **WORD** pointed to by *elements* will indicate the number of array elements used per row. In addition, *rows_used* will be filled in with actual number of rows used by the color array and the **WORD** pointed to by *status* will be filled in with 0 if the operation was successful or 1 if at least one element could not be determined. Finally, the **WORD** array (with $(num_rows * row_length)$ elements) pointed to by *colarray* will be filled in with the color index array stored one row at a time. On return *colarray* will actually contain $(elements * rows_used)$ valid elements.

BINDING `WORD i;`

```
contrl[0] = 27;  
contrl[1] = 2;  
contrl[3] = 0;  
contrl[6] = handle;  
contrl[7] = row_length;  
contrl[8] = num_rows;
```

```
ptsin[0] = pxy[0];  
ptsin[1] = pxy[1];  
ptsin[2] = pxy[2];  
ptsin[3] = pxy[3];
```

```
vdi();

*e1_used = contrl[9];
*rows_used = contrl[10];
*status = contrl[11];

for(i = 0;i < contrl[4];i++)
    colarray[i] = intout[i];
```

CAVEATS No driver types are required to utilize this function. It is therefore recommended that it be avoided unless your application is aware of the capabilities of the driver.

SEE ALSO v_cellarray()

vq_chcells()

VOID vq_chcells(*handle*, *rows*, *columns*)

WORD *handle*;

WORD **rows*, **columns*;

vq_chcells() returns the current number of columns and rows on the alpha text mode of the device.

OPCODE 5

SUB-OPCODE 1

AVAILABILITY Supported by all screen and printer drivers.

PARAMETERS *handle* specifies a valid workstation handle. *rows* and *columns* each point to a **WORD** which will be filled in with the current number of rows and columns of the device (in text mode).

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 1;
contrl[6] = handle;

vdi();

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

SEE ALSO v_curtext()

vq_color()

WORD vq_color(*handle*, *index*, *flag*, *rgb*)

WORD *handle*, *index*, *flag*;

WORD **rgb*;

vq_color() returns RGB information for a particular **VDI** color index.

OPCODE 26

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *index* specifies the **VDI** color index of which you wish to inquire. *rgb* points to an array of 3 **WORD**s which will be filled in with the red, green, and blue values (0-1000) of the color index. The values returned in the RGB array are affected by the value of *flag* as follows:

Name	<i>flag</i>	Values returned in <i>rgb</i>
COLOR_REQUESTE D	0	Return the values as last requested by the user (ie: not mapped to the actual color value displayed).
COLOR_ACTUAL	1	Return the values as the actual color being displayed.

BINDING

```

contrl[0] = 26;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = index;
intin[1] = flag;

vdi();

rgb[0] = intout[1];
rgb[1] = intout[2];
rgb[2] = intout[3];

return intout[0];

```

RETURN VALUE **vq_color()** returns -1 if the specified index is out of range for the device.

COMMENTS

Some drivers for color printers do not allow you to modify the color of each register. A simple test will allow you to determine if the driver will allow you to change index colors as follows:

- Call **vq_color()** with a *flag* value of 0 and save the return.
- Call **vs_color()** to modify that color index by a significant value.
- Call **vq_color()** with a *flag* value of 0 and compare with what you set.
- Restore the old value.

- If equivalent values are returned, you may modify each color index.

SEE ALSO `vs_color()`

vq_curaddress()

VOID `vq_curaddress(handle, row, column)`

WORD `handle`;

WORD `*row, *column`;

`vq_curaddress()` returns the current position of the alpha text cursor.

OPCODE 5

SUB-OPCODE 15

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. The **WORDS** pointed to by *row* and *column* will be filled in with the current row and column respectively of the text cursor in alpha mode.

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 15;
contrl[6] = handle;

vdi();

*row = intout[0];
*column = intout[1];
```

SEE ALSO `v_curtext()`, `vq_chcells()`

vq_extnd()

VOID `vq_extnd(handle, mode, work_out)`

WORD `handle, mode`;

WORD `*work_out`;

`vq_extnd()` returns extra information about a particular workstation.

OPCODE 102

7.90 – VDI/GDOS Function Reference

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. If *mode* is set to 0 then this call fills in the array pointed to by *work_out* with the same 57 **WORDS** which are returned by either **v_opnwk()** or **v_opnvwk()**. If *mode* is 1 then the 57 **WORDS** of *work_out* are filled in with other information as follows:

<i>work_out[x]</i>	VDI Structure Member	Meaning
0	<i>screentype</i>	Type of display screen: 0 = Not screen. 1 = Separate alpha/ graphic controllers and displays. 2 = Separate alpha/ graphic controllers with common screen. 3 = Common alpha/ graphic controllers with separate image memory. 4 = Common alpha/ graphic controllers and image memory. (All known devices either return 0 or 4.)
1	<i>bgcolors</i>	Number of background colors available.
2	<i>textfx</i>	Text effects supported. (Same bitmask as with vst_effects()).
3	<i>canscale</i>	Scaling of rasters: 0 = Can't scale. 1 = Can scale.
4	<i>planes</i>	Number of planes.
5	<i>lut</i>	Lookup table supported: 0 = Table not supported. 1 = Table supported. (True color modes return a value of 0 for <i>lut</i> and >2 for <i>colors</i> in v_opnvwk()). See the caveat listed below.
6	<i>rops</i>	Performance factor. Number of 16x16 raster operations per second.
7	<i>cancontourfill</i>	v_contourfill() availability: 0 = Not available. 1 = Available.
8	<i>textrot</i>	Character rotation capability: 0 = None. 1 = 90 degree increments. 2 = Any angle of rotation.
9	<i>writemodes</i>	Number of writing modes available.
10	<i>inputmodes</i>	Highest level of input modes available: 0 = None. 1 = Request. 2 = Sample.
11	<i>textalign</i>	Text alignment capability flag: 0 = Not available. 1 = Available.
12	<i>inking</i>	Inking capability flag. 0 = Device can't ink. 1 = Device can ink.

13	<i>rubberbanding</i>	Rubberbanding capability flag: 0 = No rubberbanding. 1 = Rubberbanded lines. 2 = Rubberbanded lines and rectangles.
14	<i>maxvertices</i>	Maximum vertices for polyline, polymarker, or filled area (-1 = no maximum).
15	<i>maxintin</i>	Maximum length of intin array (-1 = no maximum).
16	<i>mousebuttons</i>	Number of mouse buttons.
17	<i>widestyles</i>	Styles available for wide lines? 0 = No 1 = Yes
18	<i>widemodes</i>	Writing modes available for wide lines? 0 = No 1 = Yes
19-56	<i>reserved1</i>	Reserved for future use.

BINDING

```
WORD i;

contrl[0] = 102;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = mode;

vdi();

for(i = 0; i < 45; i++)
    work_out[i] = intout[i];

for(i = 0; i < 13; i++)
    work_out[45+i] = ptsout[i];
```

COMMENTS

See the entry for **V_Opnwkw()** and **V_Opnvwkw()** to see how the **vq_extnd()** information and **v_opn/v/wk()** calls are integrated into a ‘C’ style structure.

CAVEATS

The *lut* member of the **VDIWORK** structure was originally misdocumented by Atari with the values reversed. The Falcon030 as well as some third-party true-color boards return the correct values. Some older boards may not, however.

One alternative method of determining if the current screen is not using a software color lookup table (i.e. true color) is to compare the value for $2 \wedge \textit{planes}$ with the number of colors in the palette found in *colors*. If this number is different, the **VDI** is not using a software color lookup table.

SEE ALSO

v_opnwkw(), **v_opnvwkw()**, **V_Opnwkw()**, **V_Opnvwkw()**

vq_gdos()

ULONG vq_gdos(VOID)

vq_gdos() determines the availability and type of **GDOS** present.

OPCODE N/A

AVAILABILITY Supported in ROM by all Atari computers.

BINDING

```
; Correct binding for vq_gdos. Some compilers
; use the name vq_vgdos for the new version
; and vq_gdos for the old version which
; looked like:
;           move.w      #-2,d0
;           trap        #2
;           cmp.w       #-2,d0
;           sne         d0
;           ext.w       d0

_vq_gdos:

move.w     #-2,d0
trap      #2
rts
```

RETURN VALUE Currently one of the following values are returned:

Name	Value	GDOS Type
GDOS_NONE	-2	GDOS not installed.
—	Any other value.	GDOS 1.0, 1.1, or 1.2 installed.
GDOS_FNT	0x5F464E54 ('_FNT')	FONTGDOS installed.
GDOS_FSM	0x5F46534D ('_FSM')	FSMGDOS installed.

COMMENTS Calling a **GDOS** function without **GDOS** loaded is fatal and will cause a system crash.

To determine whether **FSMGDOS** or **SpeedoGDOS** is loaded look for the 'FSMC' cookie in the cookie jar. The cookie value points to a longword which will contain either '_FSM' or '_SPD'.

vq_key_s()

VOID vq_key_s(*handle*, *status*)

WORD *handle*;

WORD **status*;

vq_key_s() returns the current shift-key status.

OPCODE 128

AVAILABILITY Supported by all Atari computers.

PARAMETERS *handle* specifies a valid workstation handle. *status* points to a **WORD** which is filled in on function exit with a bit mask containing the current shift key status as follows:

Name	Bit	Meaning
K_RSHIFT	0	Right shift key depressed
K_LSHIFT	1	Left shift key depressed
K_CTRL	2	Control key depressed
K_ALT	3	Alternate key depressed

```
BINDING      contrl[0] = 128;
              contrl[1] = contrl[3] = 0;
              contrl[6] = handle;

              vdi();

              *status = intout[0];
```

SEE ALSO [graf_mkstate\(\)](#)

vq_mouse()

VOID vq_mouse(*handle*, *mb*, *mx*, *my*)

WORD *handle*;

WORD **mb*, **mx*, **my*;

vq_mouse() returns information regarding the current state of the mouse.

OPCODE 124

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. *mb* points to a **WORD** which will be filled in upon function exit with a bit mask indicating the current status of the mouse buttons as follows:

Name	Mask	Meaning
LEFT_BUTTON	0x01	Left mouse button
RIGHT_BUTTON	0x02	Right mouse button
MIDDLE_BUTTON	0x04	Middle button (this button would be the first button to the left of the rightmost button on the device).
—	0x08 . .	Other buttons (0x08 is the mask for the button to the immediate left of the middle button. Masks continue leftwards).

mx and *my* both point to **WORD**s which will be filled in upon function exit with the current position of the mouse pointer.

BINDING

```
contrl[0] = 124;  
contrl[1] = contrl[3] = 0;  
contrl[6] = handle;
```

```
vdi();
```

```
*mb = intout[0];  
*mx = ptsout[0];  
*my = ptsout[1];
```

SEE ALSO

graf_mkstate(), **v_key_s()**

vq_scan()

VOID vq_scan(*handle*, *grh*, *passes*, *alh*, *apage*, *div*)

WORD *handle*;

WORD **grh*, **passes*, **alh*, **apage*, **div*;

vq_scan() returns information regarding printer banding.

OPCODE

5

SUB-OPCODE

24

AVAILABILITY

Supported by all printer drivers.

PARAMETERS

handle specifies a valid workstation handle. *passes* specifies the number of graphic passes per printer page.

The value obtained through the formula *grh/div* specifies the number of graphics scan lines per pass. The value obtained by the formula *alh/div* specifies the number of graphic scan lines per alpha text line. *apage* specifies the number of alpha lines per page.

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 24;
contrl[6] = handle;

vdi();

*grh = intout[0];
*passes = intout[1];
*alh = intout[2];
*apage = intout[3];
*div = intout[4];
```

COMMENTS This call has been previously mis-documented.

vq_tabstatus()

WORD vq_tabstatus(*handle*)

WORD *handle*;

vq_tabstatus() determines the availability of a tablet device.

OPCODE 5

SUB-OPCODE 16

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 16;
contrl[6] = handle;

vdi();

return intout[0];
```

RETURN VALUE **vq_tabstatus()** returns 0 if no tablet is available or 1 if a tablet device is present.

SEE ALSO **vq_tdimensions(), vt_origin(), vt_axis(), vt_resolution(), vt_alignment()**

vq_tdimensions()

VOID vq_tdimensions(*handle*, *xdim*, *ydim*)
WORD *handle*;
WORD **xdim*, **ydim*;

vq_tdimensions() returns the scanning dimensions of the attached graphics tablet.

OPCODE 5

SUB-OPCODE 84

AVAILABILITY Supported by all tablet drivers.

PARAMETERS *handle* specifies a valid workstation handle. *xdim* and *ydim* point to **WORD**s which upon function exit will contain the X and Y dimensions of the tablet scanning area specified in tenths of an inch.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 84;  
contrl[6] = handle;  
  
vdi();  
  
*xdim = intout[0];  
*ydim = intout[1];
```

SEE ALSO vq_tabstatus()

vqf_attributes()

VOID vqf_attributes(*handle*, *attr*)
WORD *handle*;
WORD **attr*;

vqf_attributes() returns information regarding the current fill attributes.

OPCODE 37

AVAILABILITY Supported by all devices.

PARAMETERS *handle* specifies a valid workstation handle. *attr* points to an array of five **WORD**s which upon exit will be filled in as follows:

<i>attr[x]</i>	Meaning
0	Current fill area interior type (see <code>vsf_interior()</code>).
1	Current fill area color (see <code>vsf_color()</code>).
2	Current fill area style (see <code>vsf_style()</code>).
3	Current writing mode (see <code>vswr_mode()</code>).
4	Current perimeter status (see <code>vsf_perimeter()</code>).

BINDING

```

contrl[0] = 37;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;

vdi();

attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];

```

SEE ALSO

`vqt_attributes()`, `vql_attributes()`, `vqm_attributes()`

vqin_mode()

VOID `vqin_mode(handle, dev, mode)`

WORD *handle, dev*;

WORD **mode*;

`vqin_mode()` returns the input status of the specified **VDI** device.

OPCODE

115

AVAILABILITY

Supported by all Atari computers.

PARAMETERS

handle specifies a valid workstation handle. *mode* points to a **WORD** which upon exit will be filled in with 1 if the specified device is in request mode or 2 if in sample mode. *dev* specifies the device to inquire as follows:

Name	<i>dev</i>	Device
LOCATOR	1	Locator (Mouse, Mouse Buttons, and Keyboard)
VALUATOR	2	Valuator (not currently defined)
CHOICE	3	Choice (not currently defined)
STRING	4	String (Keyboard)

BINDING

```
contrl[0] = 115;
```

```
contrl[1] = 0
contrl[3] = 1;
contrl[6] = handle;

intin[0] = dev;

vdi();

*mode = intout[0];
```

SEE ALSO **vsin_mode()**

vql_attributes()

VOID vql_attributes(*handle*, *attr*)

WORD *handle*;

WORD **attr*;

vql_attributes() returns information regarding current settings which affects line drawing functions.

OPCODE 36

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *attr* is an array of 6 **WORDS** which describe the current parameters for line drawing as follows:

<i>attr[x]</i>	Meaning
0	Line type (see vsl_type()).
1	Line color (see vsl_color()).
2	Writing mode (see vswr_mode()).
3	End style for start of lines (see vsl_ends()).
4	End style for end of lines (see vsl_ends()).
5	Current line width (see vsl_width()).

BINDING

```
contrl[0] = 36;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;

vdi();

attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];
```

```
attr[5] = intout[5];
```

SEE ALSO vqm_attributes(), vqt_attributes(), vqf_attributes()

vqm_attributes()

VOID vqm_attributes(*handle*, *attr*)

WORD *handle*;

WORD **attr*;

vqm_attributes() returns information regarding current settings which apply to polymarker output.

OPCODE 36

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *attr* points to an array of 5 **WORDS** which specify the current polymarker attributes as follows:

<i>attr[x]</i>	Meaning
0	Marker type (see vsm_type()).
1	Marker color (see vsm_color()).
2	Writing mode (see vswr_mode()).
3	Polymarker width (see vsm_height()).
4	Polymarker height (see vsm_height()).

BINDING

```
contrl[0] = 36;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;
```

```
vdi();
```

```
attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];
```

SEE ALSO vql_attributes(), vqt_attributes(), vqf_attributes()

vqp_error()

WORD vqp_error(*handle*)

WORD *handle*;

vqp_error() returns error information for the camera driver.

OPCODE 5

SUB-OPCODE 96

AVAILABILITY Supported by all camera drivers.

PARAMETERS *handle* specifies a valid workstation handle.

BINDING

```
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 96;  
contrl[6] = handle;
```

```
vdi();
```

```
return intout[0];
```

RETURN VALUE **vqp_error()** returns the current error state as follows:

Return Value	Error State
0	No error.
1	Open dark slide for print film.
2	No port at location specified by driver.
3	Palette not found at specified port.
4	Video cable disconnected.
5	Memory allocation error.
6	Inadequate memory for buffer.
7	Memory not freed.
8	Driver file not found.
9	Driver file is not correct type.
10	Prompt user to process print film.

COMMENTS Use of this function does not stop the generation of on-screen messages. You must use **vsp_message()** to accomplish that.

SEE ALSO **vsp_message()**

vqp_films()

VOID vqp_films(*handle*, *films*)

WORD *handle*;

char **films*;

vqp_films() returns strings which represent up to five possible film types for the camera driver to utilize.

OPCODE 5

SUB-OPCODE 91

AVAILABILITY Supported by all camera drivers.

PARAMETERS *handle* specifies a valid workstation handle. *films* is a character pointer to a buffer at least 125 characters in length. Upon return *films* will be filled in with 5 character strings. Bytes 0-24 will contain a string for the first type of film, bytes 25-49 will contain a string for the second type, and so on. These strings are **not** NULL-terminated but are padded with spaces.

BINDING

```
WORD i;

contrl[0] = 5;
contrl[1] = contrl[3] = 0;
contrl[5] = 91;
contrl[6] = handle;

vdi();

for(i = 0; i < 125; i++)
    films[i] = (char)intout[i];
```

SEE ALSO **vqp_state()**

vqp_state()

VOID vqp_state(*handle*, *port*, *film*, *lightness*, *interlace*, *planes*, *indices*)

WORD *handle*;

WORD **port*, **film*, **lightness*, **interlace*, **planes*, **indices*;

vqp_state() returns information regarding the current state of the palette driver.

OPCODE 5

SUB-OPCODE 92

AVAILABILITY Supported by all camera drivers.

PARAMETERS *handle* specifies a valid workstation handle. The rest of the parameters are all **WORD**s which are filled in as follows:

Parameter	Meaning
<i>port</i>	Communication port number.
<i>film</i>	Film type (0 – 4).
<i>lightness</i>	Lightness (-3 – 3). A value of 0 specifies the current f-stop setting. A value of three results in an exposure half as long as normal while a value of 3 results in an exposure twice as long as normal.
<i>interlace</i>	Interlace mode. A value of 0 is non-interlaced, 1 is interlaced.
<i>planes</i>	Number of planes (1 – 4)
<i>indices</i>	This is actually a WORD array with at least 16 members. ($2 \wedge \text{planes}$) members will be filled in with color codes for the driver. <i>indices[0]</i> and <i>indices[1]</i> will specify the first color, <i>indices[2]</i> and <i>indices[2]</i> the second, and so on.

BINDING

```
WORD i;  
  
contrl[0] = 5;  
contrl[1] = contrl[3] = 0;  
contrl[5] = 92;  
contrl[6] = handle;  
  
vdi();  
  
*port = intout[0];  
*film = intout[1];  
*lightness = intout[2];  
*interlace = intout[3];  
*planes = intout[4];  
  
for(i = 0; i < 21; i++)  
    indices[i] = intout[5 + i];
```

SEE ALSO `vsp_state()`

vqt_advance()

VOID `vqt_advance(handle, wch, advx, advy, xrem, yrem)`

WORD *handle, wch*;

WORD **advx, *advy, *xrem, *yrem*;

`vqt_advance()` returns the advance vector and remainder for a character.

OPCODE	247
AVAILABILITY	Available only with FSMGDOS or SpeedoGDOS .
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>wch</i> contains the character which you desire information for. Upon return the WORD s pointed to by <i>advx</i> , <i>advy</i> , <i>xrem</i> , and <i>yrem</i> will be filled in with the correct advance vector and remainders.
BINDING	<pre>contrl[0] = 247; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; intin[0] = wch; vdi(); *advx = ptsout[0]; *advy = ptsout[1]; *xrem = ptsout[2]; *yrem = ptsout[3];</pre>
COMMENTS	<i>advx</i> and <i>advy</i> , when added to the position where the character was rendered will indicate the position to draw the next character. This advance vector works in all directions with all character rotations. <i>xrem</i> and <i>yrem</i> give the remainder value as a modulus of 16384. These remainders should be summed by an application an managed to nudge the advance vector by a pixel when necessary.
SEE ALSO	vqt_width() , vqt_extent() , vqt_f_extent()

vqt_advance32()

VOID vqt_advance32(*handle*, *wch*, *advx*, *advy*)

WORD *handle*, *wch*;

fix31 **advx*, **advy*;

vqt_advance32() is a variation of the binding for **vqt_advance()** which returns the advance vector and remainder for a character as two **fix31** values..

OPCODE 247

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *wch* contains the character which you desire information for. Upon return the **fix31**s pointed to by *advx* and *advy* will be filled in with the correct advance vector.

BINDING

```
contrl[0] = 247;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = wch;

vdi();

*advx = (fix31)((ptsout[4] << 16) | ptsout[5]);
*advy = (fix31)((ptsout[6] << 16) | ptsout[7]);
```

COMMENTS *advx* and *advy*, when added to the position where the character was rendered will indicate the position to draw the next character. This advance vector works in all directions with all character rotations.

SEE ALSO [vqt_width\(\)](#), [vqt_extent\(\)](#), [vqt_f_extent\(\)](#)

vqt_attributes()

VOID [vqt_attributes\(\)](#) (*handle*, *attr*)

WORD *handle*;

WORD **attr*;

[vqt_attributes\(\)](#) returns information regarding the current attributes which affect text output.

OPCODE 38

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *attr* points to an array containing 10 **WORDS** which are filled in upon function exit as follows:

<i>attr[x]</i>	Meaning
0	Text face (see vst_font()).
1	Text color (see vst_color()).
2	Text rotation (see vst_rotation()).
3	Horizontal alignment (see vst_alignment()).
4	Vertical alignment (see vst_alignment()).
5	Writing mode (see vswr_mode()).
6	Character width (see vst_height()).
7	Character height (see vst_height()).
8	Character cell width (see vst_height()).
9	Character cell height (see vst_height()).

BINDING

```

contrl[0] = 38;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;

vdi();

attr[0] = intout[0];
attr[1] = intout[1];
attr[2] = intout[2];
attr[3] = intout[3];
attr[4] = intout[4];
attr[5] = intout[5];
attr[6] = intout[6];
attr[7] = intout[7];
attr[8] = intout[8];
attr[9] = intout[9];

```

COMMENTS The values pertaining to character and cell width and have limited usefulness as they are only constant with non-proportional fonts.

SEE ALSO `vql_attributes()`, `vqm_attributes()`, `vqf_attributes()`

vqt_cachesize()

WORD `vqt_cachesize(handle, which, size)`

WORD `handle, which;`

LONG `*size;`

`vqt_cachesize()` returns the size of the largest allocatable block of memory in one of two caches.

OPCODE 255

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *which* specifies which cache. A value of **CACHE_CHAR** (0) selects the character bitmap cache. A value of **CACHE_MISC** (1) selects the miscellaneous cache. The **LONG** pointed to by *size* will be filled in upon function exit with the size of the largest allocatable block of memory in the selected cache.

BINDING

```

contrl[0] = 255;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = which;

vdi();

```

```
*size = (LONG)(((LONG)intin[0] << 16) | (LONG)intin[1]);
```

COMMENTS

An application can estimate the amount of memory required to generate a character and print a warning message if the user attempts to exceed it. **FSMGDOS** will simply print a message on screen (you can intercept this with **vst_error()**) and ask the user to reboot. You can estimate the amount of memory required for a particular character in the character bitmap cache with the formula:

$$(\text{width in pixels} + 7)/8 * \text{height in pixels}$$

Likewise, you can estimate the amount of memory needed for the miscellaneous cache as:

$$84 * (\text{width} + \text{height})$$

SEE ALSO

vst_error(), **v_flushcache()**

vqt_devinfo()

VOID vqt_devinfo(*handle*, *devid*, *exists*, *devstr*)

WORD *handle*, *devid*;

WORD **exists*;

char **devstr*;

vqt_devinfo() determines if a particular device ID is available, and if so, the name of the device driver.

OPCODE

248

AVAILABILITY

Available only with **FONTGDOS**, **FSM**, or **SpeedoGDOS**.

PARAMETERS

handle specifies a valid workstation handle. *devid* specifies the device ID as listed in the 'ASSIGN.SYS' file. *exists* is a pointer to a **WORD** which will be filled in with **DEV_INSTALLED** (1) if a device is installed with the specified ID number or **DEV_MISSING** (0) if not. If the device does exist, the character buffer pointer to by *devstr* will be filled in with the filename of the device padded with spaces to the standard **GEMDOS** 8 + 3 format.

BINDING

WORD *i*;

```
contrl[0] = 248;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;
```

```
intin[0] = devid;
```

```

vdi();

*exists = ptsout[0];

for(i = 0;i < contrl[4];i++)
    devstr[i] = (char)intout[i];

```

vqt_extent()

VOID vqt_extent(*handle*, *str*, *pts*)

WORD *handle*;

char **str*;

WORD **pts*;

vqt_extent() returns the pixel extent of a string of text.

OPCODE 116

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *str* points to a text string to return extent information for. *pts* points to an array of 8 **WORD**s which will be filled in as follows:



<i>pts[x]</i>	Meaning
0	X coordinate of point 1.
1	Y coordinate of point 1.
2	X coordinate of point 2.
3	Y coordinate of point 2.
4	X coordinate of point 3.
5	Y coordinate of point 3.
6	X coordinate of point 4.
7	Y coordinate of point 4.

BINDING

```

WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 116;
contrl[1] = 0;
contrl[3] = --i;

```

```
contrl[6] = handle;

vdi();

pts[0] = ptsout[0];
pts[1] = ptsout[1];
pts[2] = ptsout[2];
pts[3] = ptsout[3];
pts[4] = ptsout[4];
pts[5] = ptsout[5];
pts[6] = ptsout[6];
pts[7] = ptsout[7];
```

COMMENTS This function will also output correct bounding information for rotated text. It is recommended that **vqt_f_extent()** be used for outline fonts as it takes special factors into consideration which makes its output more accurate.

SEE ALSO **vqt_f_extent()**, **vqt_advance()**, **vqt_width()**

vqt_f_extent()

VOID vqt_f_extent(*handle*, *str*, *pts*)

WORD *handle*;

char **str*;

WORD **pts*;

vqt_f_extent() returns the bounding box required to enclose the specified string of text.

OPCODE 240

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS Same as **vqt_extent()**.

BINDING

```
WORD i = 0;

while(intin[i++] = (WORD)*str++);

contrl[0] = 240;
contrl[1] = 0;
contrl[3] = --i;
contrl[6] = handle;

vdi();

pts[0] = ptsout[0];
pts[1] = ptsout[1];
pts[2] = ptsout[2];
pts[3] = ptsout[3];
pts[4] = ptsout[4];
```

```
pts[5] = ptsout[5];
pts[6] = ptsout[6];
pts[7] = ptsout[7];
```

COMMENTS As opposed to `vqt_extent()`, `vqt_f_extent()` calculates the remainders generated by outline fonts therefore providing more accurate results.

SEE ALSO `vqt_extent()`, `vqt_width()`, `vqt_advance()`

vqt_f_extent16()

VOID `vqt_f_extent(handle, wstr, wstrlen, pts)`

WORD `handle`;

WORD `*wstr`;

WORD `wstrlen`;

WORD `*pts`;

`vqt_f_extent16()` is a variant binding of `vqt_f_extent()` that returns the bounding box required to enclose the specified string of 16-bit Speedo character indexed text.

OPCODE 240

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS `handle` specifies a valid workstation handle. `wstr` points to a 16-bit text string composed of Speedo character indexes. `wstrlen` indicates the length of `wstr`. The array pointed to by `pts` is filled in with the same values as `vqt_extent()`.

BINDING WORD `i`;

```
for( i = 0; i < wstrlen; i++)
    intin[i] = wstr[i];
```

```
contrl[0] = 240;
contrl[1] = 0;
contrl[3] = wstrlen;
contrl[6] = handle;
```

```
vdi();
```

```
pts[0] = ptsout[0];
pts[1] = ptsout[1];
pts[2] = ptsout[2];
pts[3] = ptsout[3];
pts[4] = ptsout[4];
pts[5] = ptsout[5];
pts[6] = ptsout[6];
pts[7] = ptsout[7];
```

COMMENTS This variation of the `vqt_f_extent()` binding should only be used when **SpeedoGDOS** has been properly configured with `vst_charmap()`.

SEE ALSO `vqt_extent()`, `vqt_width()`, `vqt_advance()`

vqt_fonthead()

VOID `vqt_fonthead(handle, buffer, pathname)`

WORD `*handle;`

char `*buffer, *pathname;`

`vqt_fonthead()` returns font-specific information for the currently selected Speedo font.

OPCODE 234

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *buffer* should point to a buffer of at least 421 bytes into which the font header will be copied. *pathname* should point to a buffer of at least 128 bytes into which the full pathname of the font's corresponding '.TDF' file will be copied.

BINDING

WORD *i*;

```
contrl[0] = 234;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;
```

```
vdi();
```

```
for(i = 0; i < contrl[4]; i++)
    pathname[i] = (char)intout[i];
```

COMMENTS The font header format and '.TDF' file contents are contained in *Appendix G: Speedo Fonts*.

SEE ALSO `vqt_fontinfo()`

vqt_fontinfo()

VOID vqt_fontinfo(*handle*, *first*, *last*, *dist*, *width*, *effects*)

WORD *handle*;

WORD **first*, **last*, **dist*, **width*, **effects*;

vqt_fontinfo() returns information regarding the current text font.

OPCODE 131

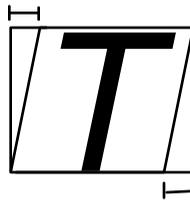
AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *first* and *last* each point to a **WORD** which will be filled in with the first and last character in the font respectively. *dist* points to an array of 5 **WORD**s which indicate the distances between the baseline and the point indicated as follows:



width specifies the width of the largest cell in the font in pixels not including effects. *effects* points to an array of 3 **WORD**s which contain information relating to the offsets of the font when printed with the current effects.

effects[0]



effects[1]

$effects[2] = effects[0] + effects[1]$

effects[0] specifies the number of X pixels of the left slant. *effects[1]* specifies the number of X pixels of the right slant. *effects[2]* specifies the extra number of X

pixels to add to compensate for the special effects.

BINDING

```
contrl[0] = 131;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;

vdi();

*first = intout[0];
*last = intout[1];
*width = ptsout[0];
dist[0] = ptsout[1];
dist[1] = ptsout[3];
dist[2] = ptsout[5];
dist[3] = ptsout[7];
effects[0] = ptsout[2];
effects[1] = ptsout[4];
effects[2] = ptsout[6];
```

CAVEATS

SpeedoGDOS is not capable of generating values for *dist[1]* or *dist[2]* so *dist[1]* is set to equal *dist[0]* and *dist[2]* is set to equal *dist[3]*.

SEE ALSO

`vqt_width()`

vqt_get_table()

VOID vqt_get_table(*handle*, *map*)

WORD *handle*;

VOID *******map*;

vqt_get_table() returns pointers to seven tables which map the Atari character set to the Bitstream character indexes.

OPCODE

254

AVAILABILITY

Available only with **SpeedoGDOS**.

PARAMETERS

handle specifies a valid workstation handle. The location pointed to by *map* will be filled in with a pointer to seven internal tables, each 224 **WORD** size entries long mapping ASCII characters 32–255 to Bitstream character indexes.

The tables are defined as follows:

Position	Table
1st	Master mapping.
2nd	Bitstream International Character Set
3rd	Bitstream International Symbol Set

4th	Bitstream Dingbats Set
5th	PostScript Text Set
6th	PostScript Symbol Set
7th	PostScript Dingbats Set

BINDING

```

contrl[0] = 254;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;

vdi();

*(VOID *)map = ((LONG)(intout[0] << 16) | (LONG)intout[1]);

```

COMMENTS

Use of this call allows access to characters outside of the ASCII range but care must be taken to as this call affects all applications.

vqt_name()

WORD vqt_name(*handle*, *index*, *fontname*)

WORD *handle*;

WORD *index*;

char **fontname*;

vqt_name() returns the name of the specified font.

OPCODE

130

AVAILABILITY

Supported by all drivers.

PARAMETERS

handle specifies a valid workstation handle. *fontname* points to a character buffer of at least 33 characters which will be filled in with the name of font *index* and a flag which distinguishes bitmap and outline fonts. *fontname*[0–31] will contain the name of the font (not necessarily **NULL**-terminated).

If **FSMGDOS** or **SpeedoGDOS** is installed, *fontname*[32] will contain a flag equalling **OUTLINE_FONT** (1) if the specified font is an outline font or **BITMAP_FONT** (0) if it is a bitmap font.

BINDING

```

WORD i;

contrl[0] = 130;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = index;

vdi();

```

```
for(i = 0; i < 33; i++)
    fontname[i] = intout[i + 1];

return intout[0];
```

RETURN VALUE `vqt_name()` returns the unique code value which identifies this font (and is passed to `vst_font()`).

SEE ALSO `vst_load_fonts()`, `vst_font()`

vqt_pairkern()

VOID `vqt_pairkern(handle, char1, char2, x, y)`

WORD `char1, char2;`

fix31 `*x, *y;`

`vqt_pairkern()` returns adjustment vector information for the kerning of a character pair.

OPCODE 235

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *char1* and *char2* specify the left and right members of the character pair to inquire. *x* and *y* will be filled with the adjustment vector for the specified character pair.

BINDING

```
contrl[0] = 235;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = char1;
intin[1] = char2;

vdi();

*x = ((LONG)ptsout[0] << 16 ) | ptsout[1];
*y = ((LONG)ptsout[2] << 16 ) | ptsout[3];
```

SEE ALSO `vqt_trackkern()`, `vst_kern()`

vqt_trackkern()

VOID vqt_trackkern(*handle*, *x*, *y*)

fix31 **x*, **y*;

vqt_trackkern() returns the horizontal and vertical adjustment vector for track kerning.

OPCODE 234

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *x* and *y* are the horizontal and vertical adjustment vectors currently used to modify character spacing in track kerning.

BINDING

```
contrl[0] = 234;  
contrl[1] = 0;  
contrl[3] = 0;  
contrl[6] = handle;
```

```
vdi();
```

```
*x = ((LONG)ptsout[0] << 16 ) | ptsout[1];  
*y = ((LONG)ptsout[2] << 16 ) | ptsout[2];
```

SEE ALSO **vqt_pairkern()**, **vst_kern()**

vqt_width()

WORD vqt_width(*handle*, *wch*, *cellw*, *left*, *right*)

WORD *handle*, *wch*;

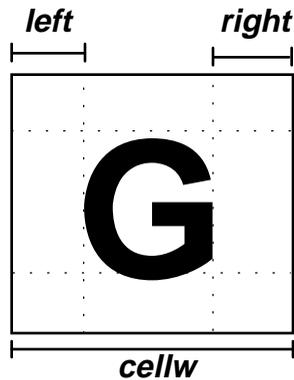
WORD **cellw*, **left*, **right*;

vqt_width() returns information regarding the width of a character cell.

OPCODE 117

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. The lower eight bits of *wch* specify the ASCII character to return width information about. The following three values are each WORDs which are filled in by the function upon return with information about the width of the specified character in pixels as illustrated here.

**BINDING**

```
contrl[0] = 117;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;  
  
intin[0] = wch;  
  
vdi();  
  
*cellw = ptsout[0];  
*left = ptsout[2];  
*right = ptsout[4];  
  
return intout[0];
```

RETURN VALUE

vqt_width() returns *wch* or -1 if an error occurred.

CAVEATS

vqt_width() does not take into account remainders when dealing with outline fonts. It is therefore recommended that **vqt_advance()** be used instead when inquiring about outline fonts.

SEE ALSO

vqt_advance()

vr_recfl()

VOID vr_recfl(*handle*, *pxy*)

WORD *handle*;

WORD **pxy*;

vr_recfl() outputs a filled rectangle.

OPCODE 114

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *pxy* points to an array of 4 **WORDS** which give a **VDI** format rectangle of the object to draw.

BINDING

```
contrl[0] = 114;  
contrl[1] = 2;  
contrl[3] = 0;  
contrl[6] = handle;  
  
ptsin[0] = pxy[0];  
ptsin[1] = pxy[1];  
ptsin[2] = pxy[2];  
ptsin[3] = pxy[3];  
  
vdi();
```

COMMENTS **vr_recfl()**, as opposed to **v_bar()**, never draws an outline regardless of the settings of **vsf_perimeter()**.

SEE ALSO **v_bar()**

vr_trnfm()

VOID vr_trnfm(*handle*, *src*, *dest*)

WORD *handle*;

MFDB **src*, **dest*;

vr_trnfm() transforms a memory block from device-independent to device-dependent and vice-versa.

OPCODE 110

AVAILABILITY Supported by all drivers.

7.118 – VDI/GDOS Function Reference

PARAMETERS *handle* specifies a valid workstation handle. *src* specifies the **MFDB** (as defined in `vro_cpyfm()`) whereas *dest* specifies the **MFDB** of the destination.

BINDING

```
contrl[0] = 110;
contrl[1] = contrl[3] = 0;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)src >> 16);
contrl[8] = (WORD)src;
contrl[9] = (WORD)((LONG)dest >> 16);
contrl[10] = (WORD)dest;

vdi();
```

CAVEATS While `vr_trnfm()` will work for in-place transformations, this process can be time-consuming for large forms.

This call will not translate between forms with multiple planes. For instance, you can not translate a 2 plane device-independent image to an 8-plane device-specific image.

COMMENTS To stay compatible with future hardware developments it is recommended that all images be initially either stored or manually translated to device-independent format and subsequently converted with this function to match the planar configuration of the device.

When this call is used to transform forms with either 2 or 4 bit planes, color translation is performed on each pixel as follows:

Four-Plane Transformations

Device	VDI
0000	0
0001	2
0010	3
0011	6
0100	4
0101	7
0110	5
0111	8

Device	VDI
1000	9
1001	10
1010	11
1011	14
1100	12
1101	15
1110	13
1111	1

Two Plane

Device	VDI
00	0
01	2
10	3
11	1

SEE ALSO `vro_cpyfm()`

vro_cpyfm()

VOID vro_cpyfm(handle, mode, pxy, src, dest)

WORD handle, mode;

WORD *pxy;

MFDB *src, *dest;

vro_cpyfm() ‘blits’ a screen or memory block from one location to another.

OPCODE 109

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies valid workstation handle. *mode* specifies the writing mode as follows:

Name	Mode	Result
ALL_WHITE	0	All zeros.
S_AND_D	1	source AND destination
S_AND_NOTD	2	source AND (NOT destination)
S_ONLY	3 (Replace mode)	source
NOTS_AND_D	4 (Erase mode)	(NOT source) AND destination
D_ONLY	5	destination
S_XOR_D	6 (XOR Mode)	source XOR destination
S_OR_D	7	source OR destination
NOT_SORD	8	NOT (source OR destination)
NOT_SXORD	9	NOT (source XOR destination)
NOT_D	10	NOT destination
S_OR_NOTD	11	source OR (NOT destination)
NOT_S	12	NOT source
NOTS_OR_D	13	(NOT source) OR destination
NOT_SANDD	14	NOT (source AND destination)
ALL_BLACK	15	All ones.

pxy points to an array of eight **WORDS**. *pxy[0–3]* contains the bounding rectangle of the source block. *pxy[4–7]* contains the bounding rectangle of the destination block. *src* and *dest* each point to an **MFDB** structure which describes the source and destination memory form. **MFDB** is defined as follows:

```
typedef struct
{
```

7.120 – VDI/GDOS Function Reference

```
/* Memory address (NULL = current screen). If you specify
a value of NULL, the rest of the structure will be filled
out for you. */
VOID *fd_addr;

/* Form width in pixels */
WORD fd_width;

/* Form height in pixels */
WORD fd_height;

/* Form width in WORDs (fd_width + 15)/16 */
WORD fd_wdwidth;

/* Format (0 = device-specific, 1 = VDI format) */
WORD fd_stand;

/* Number of memory planes */
WORD fd_planes;

/* Reserved (set to 0) */
WORD reserved1;
WORD reserved2;
WORD reserved3;
} MFDB;
```

BINDING

```
contrl[0] = 109;
contrl[1] = 4;
contrl[3] = 1;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)src >> 16);
contrl[8] = (WORD)src;
contrl[9] = (WORD)((LONG)dest >> 16);
contrl[10] = (WORD)dest;

intin[0] = mode;

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];
ptsin[4] = pxy[4];
ptsin[5] = pxy[5];
ptsin[6] = pxy[6];
ptsin[7] = pxy[7];

vdi();
```

COMMENTS

To 'blit' a single-plane form to a multi-plane destination, use **vrt_cpyfm()**.

SEE ALSO

vr_trnfm(), **vrt_cpyfm()**

vrq_choice()

VOID vrq_choice(*handle*, *start*, *final*)

WORD *handle*, *start*;

WORD **final*;

vrq_choice() accepts input from the ‘choice’ device in request mode.

OPCODE 30

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS *handle* specifies a valid workstation handle. *start* indicates the starting value for the choice device (1-??). *final* points to a **WORD** which will be filled in upon exit with the results of the request.

BINDING

```
contrl[0] = 30;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = start;

vdi();

*final = intout[0];
```

COMMENTS Input is sampled until a key is pressed.

SEE ALSO **vsm_choice()**, **vsin_mode()**

vrq_locator()

VOID vrq_locator(*handle*, *mx*, *my*, *xout*, *yout*, *term*)

WORD *handle*, *mx*, *my*;

WORD **xout*, **yout*, **term*;

vrq_locator() inputs information from the ‘locator’ device in request mode.

OPCODE 28

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS *handle* specifies a valid workstation handle. To start, the mouse cursor is displayed at the location given by *mx* and *my*. When a key or mouse button is pressed, the call returns. The final location of the mouse pointer is filled into the 2 **WORD**s pointed to by *xout* and *yout*. The **WORD** pointed to by *term* is filled in with the ASCII key of the character that terminated input, 32 (0x20) if the left mouse button was struck, or 33 (0x21) if the right mouse button was struck.

BINDING

```
contrl[0] = 28;  
contrl[1] = 1;  
contrl[3] = 0;  
contrl[6] = handle;  
  
ptsin[0] = mx;  
ptsin[1] = my;  
  
vdi();  
  
*term = intout[0];  
  
*xout = ptsout[0];  
*yout = ptsout[1];
```

COMMENTS Using this function will confuse the **AES**'s mouse input functions.

SEE ALSO **vsm_locator()**, **vsin_mode()**

vrq_string()

VOID **vrq_string**(*handle*, *maxlen*, *echo*, *outxy*, *str*)

WORD *handle*, *maxlen*, *echo*;

WORD **outxy*;

char **str*;

vrq_string() waits for input from the 'string' device in request mode.

OPCODE 31

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS *handle* specifies a valid workstation handle. This call inputs characters from the keyboard into the buffer pointed to by *str* up to *maxlen* + 1 characters. If *echo* is set to 1, characters are echoed to the screen at the location given by the two **WORD**s pointed to by *outxy*. If *echo* is set to 0, no echoing is performed.

BINDING

```
WORD i;  
  
contrl[0] = 31;
```

```

contrl[1] = 1;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = maxlen;
intin[1] = echo;

ptsin[0] = outxy[0];
ptsin[1] = outxy[1];

vdi();

for(i = 0; i < contrl[4]; i++)
    str[i] = (char)intout[i];

```

- CAVEATS** The *echo* parameter is not functional. Character output is never echoed. However, *outxy* must point to valid memory space or a crash *will* occur.
- COMMENTS** Though this binding does not allow for it, if *maxlen* is specified as negative, then as many as $|maxlen| + 1$ characters will be read as keycodes rather than ASCII codes. The values in *intout* will occupy the full **WORD** rather than just the lower eight bits. A custom binding could be used to take advantage of this.
- SEE ALSO** `vsin_mode()`, `vsm_string()`

vrq_valuator()

VOID vrq_valuator(*handle*, *start*, **final*, **term*)

WORD *handle*, *start*;

WORD **final*, **term*;

vrq_valuator() accepts for input from the valuator device until a terminating character is entered in request mode.

OPCODE 29

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS *handle* specifies a valid workstation handle. *start* specifies the initial value of the valuator device (1–100). When a terminating character has been struck, the **WORD** pointed to by *final* will be filled in with the final value of the valuator and the **WORD** pointed to by *term* will be filled in with whatever ASCII character caused termination.

BINDING

```

contrl[0] = 29;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

```

```
intin[0] = start;

vdi();

*final = intout[0];
*term = intout[1];
```

COMMENTS The ‘valuator’ is typically the up and down arrow keys. Each key increments or decrements the value by 10 unless the shift key is held in which case it is incremented or decremented by 1.

SEE ALSO [vsm_valuator\(\)](#), [vsin_mode\(\)](#)

vrt_cpyfm()

VOID [vrt_cpyfm](#)(*handle*, *mode*, *pxy*, *src*, *dest*, *colors*)

WORD *handle*, *mode*;

WORD **pxy*;

MFDB **src*, **dest*;

WORD **colors*;

[vrt_cpyfm](#)() ‘blits’ a single-plane source form to a multiple-plane destination.

OPCODE 121

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. *mode* specifies the writing mode (1–4, see [vswr_mode\(\)](#)). *pxy*, *src*, and *dest* are defined the same as in [vro_cpyfm\(\)](#). *colors* points to a 2 **WORD** array which specifies the colors to apply to the ‘blitted’ image. *colors[0]* is applied to all set bits in the source image and *colors[1]* is applied to all of the cleared bits.

BINDING

```
contrl[0] = 121;
contrl[1] = 4;
contrl[3] = 3;
contrl[6] = handle;
contrl[7] = (WORD)((LONG)src >> 16);
contrl[8] = (WORD)src;
contrl[9] = (WORD)((LONG)dest >> 16);
contrl[10] = (WORD)dest;

intin[0] = mode;
intin[1] = colors[0];
intin[2] = colors[1];

ptsin[0] = pxy[0];
ptsin[1] = pxy[1];
```

```
ptsin[2] = pxy[2];
ptsin[3] = pxy[3];
ptsin[4] = pxy[4];
ptsin[5] = pxy[5];
ptsin[6] = pxy[6];
ptsin[7] = pxy[7];

vdi();
```

COMMENTS The source form must be a monoplane form.

SEE ALSO vro_cpyfm()

vs_clip()

VOID vs_clip(*handle*, *flag*, *pxy*)

WORD *handle*, *flag*;

WORD **pxy*;

vs_clip() defines the global clipping rectangle and state for the specified workstation.

OPCODE 129

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *flag* is set to **CLIP_OFF** (0) to turn off clipping or **CLIP_ON** (1) to enable clipping. If *flag* is **CLIP_ON** (1) then *pxy* should point to a 4 **WORD** array containing a **VDI** format rectangle which will serve as the clipping rectangle, otherwise, *pxy* can be **NULL**.

BINDING

```
contrl[0] = 129;
contrl[1] = 2;
contrl[3] = 1;
contrl[6] = handle;

if(intin[0] = flag) {
    ptsin[0] = pxy[0];
    ptsin[1] = pxy[1];
    ptsin[2] = pxy[2];
    ptsin[3] = pxy[3];
}

vdi();
```

COMMENTS All **VDI** calls are clipped to that workstations current clipping rectangle.

vs_color()

VOID vs_color(*handle*, *color*, *rgb*)

WORD *handle*, *color*;

WORD **rgb*;

vs_color() sets the color of a palette index.

OPCODE 14

AVAILABILITY Supported by all devices.

PARAMETERS *handle* specifies a valid workstation handle. *color* specifies the color register of the color to modify. *rgb* points to an array of three **WORD**s which contain the red, green, and blue values respectively (0–1000) which will be used to map the color index to the closest color value possible.

BINDING

```
contrl[0] = 14;  
contrl[1] = 0;  
contrl[3] = 4;  
contrl[6] = handle;
```

```
intin[0] = color;  
intin[1] = rgb[0];  
intin[2] = rgb[1];  
intin[3] = rgb[2];
```

```
vdi();
```

SEE ALSO Esetcolor(), Setcolor()

vs_curaddress()

VOID vs_curaddress(*handle*, *row*, *column*)

WORD *handle*, *row*, *column*;

vs_curaddress() sets the position of the alpha screen text cursor.

OPCODE 5

SUB-OPCODE 11

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. *row* and *column* specify the new

coordinates of the text cursor.

BINDING

```
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 11;
contrl[6] = handle;

intin[0] = row;
intin[1] = column;

vdi();
```

COMMENTS This call is equivalent to the ESC-Y VT-52 code.

SEE ALSO [vq_curaddress\(\)](#)

vs_palette()

VOID vs_palette(*handle*, *mode*)

WORD *handle*, *mode*;

vs_palette() selects a CGA palette.

OPCODE 5

SUB-OPCODE 60

AVAILABILITY This call was originally designed for use on IBM CGA-based computers. Its usefulness and availability are not guaranteed under any driver so it should thus be avoided.

PARAMETERS *handle* specifies a valid workstation handle. A *mode* value of 0 selects a palette of red, green, and blue. A *mode* value of 1 selects a palette of cyan, magenta, and white.

BINDING

```
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 1;
contrl[5] = 60;
contrl[6] = handle;

intin[0] = mode;

vdi();
```

vsc_form()

VOID vsc_form(*handle*, *newform*)

MFORM **newform*;

vsc_form() alters the appearance of the mouse pointer.

OPCODE 111

AVAILABILITY Supported by all screen drivers.

PARAMETERS *handle* specifies a valid workstation handle. *newform* points to a **MFORM** structure defined as follows:

```
typedef struct
{
    WORD mf_xhot;      /* X 'hot spot' */
    WORD mf_yhot;      /* Y 'hot spot' */
    WORD mf_nplanes;   /* Number of planes (must be 1) */
    WORD mf_fg;        /* Foreground color (should be 0) */
    WORD mf_bg;        /* Background color (should be 1) */
    WORD mf_mask[16]; /* 16 WORDs of mask */
    WORD mf_data[16]; /* 16 WORDs of data */
} MFORM;
```

BINDING

```
WORD i;

contrl[0] = 111;
contrl[1] = 0;
contrl[3] = 37;
contrl[6] = handle;

for(i = 0; i < 37; i++)
    intin[i] = ((WORD *)newform)[i];

vdi();
```

SEE ALSO **graf_mouse()**

vsf_color()

WORD vsf_color(*handle*, *color*)

WORD *handle*, *color*;

vsf_color() changes the current fill color.

OPCODE	25
AVAILABILITY	Supported by all drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>color</i> specifies the new fill color index.
BINDING	<pre> contrl[0] = handle; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; intin[0] = color; vdi(); </pre>
RETURN VALUE	vsf_color() returns the actual color set (within bounds).
SEE ALSO	vst_color() , vsm_color() , vsl_color() , vsf_attributes()

vsf_interior()

WORD vsf_interior(*handle*, *interior*)

WORD *handle*, *interior*;

vsf_interior() sets the interior type for filled objects.

OPCODE	23
AVAILABILITY	Supported by all drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>interior</i> specifies the interior type as follows:

Name	<i>interior</i>	Meaning
FIS_HOLLOW	0	Hollow interior (color index 0).
FIS_SOLID	1	Solid interior (as set by vsf_color()).
FIS_PATTERN	2	Patterned fill. (style set by vsf_style()).
FIS_HATCH	3	Hatched fill. (style set by vsf_style()).
FIS_USER	4	User-defined fill (as set by vsf_udpat()).

BINDING	<pre> contrl[0] = 23; contrl[1] = 0; contrl[3] = interior; contrl[6] = handle; </pre>
----------------	---

```
intin[0] = interior;  
vdi();
```

RETURN VALUE This call returns the color value actually set (within bounds).

SEE ALSO `vsf_style()`

`vsf_perimeter()`

WORD `vsf_perimeter(handle, flag)`

WORD *handle*, *flag*;

`vsf_perimeter()` sets whether a border will be drawn around most **VDI** objects.

OPCODE 104

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *flag* is set to **PERIMETER_OFF** (0) to turn off perimeter drawing and **PERIMETER_ON** (1) to enable it.

BINDING

```
contrl[0] = 104;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;  
  
vdi();
```

RETURN VALUE This function returns the new value of the perimeter visibility flag.

`vsf_style()`

WORD `vsf_style(handle, style)`

WORD *handle*, *style*;

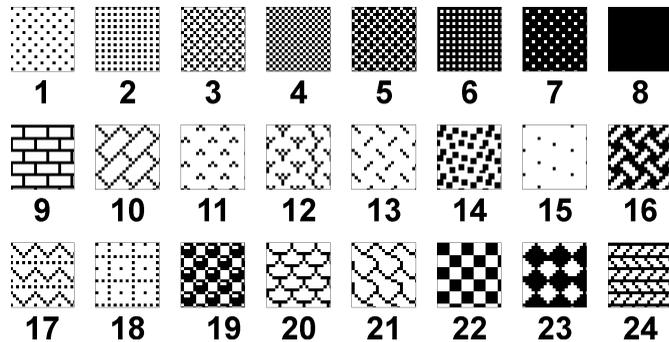
`vsf_style()` defines the style of fill pattern applied to filled objects.

OPCODE 24

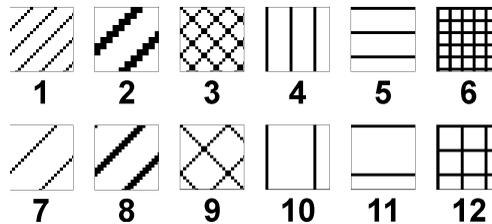
AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *style* specifies the pattern or hatch index depending upon the last setting of `vsf_interior()`. Valid pattern indexes are

as follows:



Valid hatch indexes are as follows:



BINDING

```
contrl[0] = 24;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
```

```
intin[0] = style;
```

```
vdi();
```

RETURN VALUE

This call returns the actual style set by the call.

COMMENTS

The interior type should be set first with `vsf_interior()`.

SEE ALSO

`vsf_interior()`

vsf_udpat()

VOID vsf_udpat(*handle*, *pattern*, *planes*)

WORD *handle*;

WORD **planes*;

WORD *planes*;

vsf_udpat() creates the user-defined fill pattern.

OPCODE 112

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. In palette-based modes, *pattern* points to an array of (16 * *planes*) **WORD**s which provide the bit pattern for the fill.

In true-color modes, *pattern* points to a 16x16 array of **LONG**s (256 in total) which each contain 32-bit color information. *planes* specifies the number of color planes for the fill. Use 1 for a monochrome fill on any display, a value equal to the number of planes on the current device for a palette-based color fill or 32 for a true-color display.

BINDING

```
WORD i;  
  
contrl[0] = 112;  
contrl[1] = 0;  
contrl[3] = (16 * planes);  
contrl[6] = handle;  
  
for(i = 0; i < (16 * planes); i++)  
    intin[i] = pattern[i];  
  
vdi();
```

SEE ALSO vsf_interior()

vsin_mode()

WORD vsin_mode(*handle*, *device*, *mode*)

WORD *handle*, *device*, *mode*;

vsin_mode() chooses between request or sample mode for the specified device.

OPCODE 33

AVAILABILITY Supported in ROM by all Atari computers.

PARAMETERS *handle* specifies a valid workstation handle. A *mode* value of **REQUEST_MODE** (1) sets the device to operate in request mode whereas a value of **SAMPLE_MODE** (2) operates the device in sample mode. Valid devices are:

Name	<i>device</i>	Device
LOCATOR	1	Locator
VALUATOR	2	Valuator
CHOICE	3	Choice
STRING	4	String

```
BINDING      contrl[0] = 33;
                contrl[1] = 0;
                contrl[3] = 2;
                contrl[6] = handle;

                intin[0] = device;
                intin[1] = mode;

                vdi();

                return intout[0];
```

RETURN VALUE **vsin_mode()** returns *mode*.

COMMENTS Using this function will cause the **AES** to function improperly.

SEE ALSO **vrq_valuator()**, **vrq_string()**, **vrq_choice()**, **vrq_locator()**, **vsm_valuator()**, **vsm_string()**, **vsm_choice()**, **vsm_locator()**

vsl_color()

WORD **vsl_color(*handle*, *color*)**

WORD *handle*, *color*;

vsl_color() sets the color for line-drawing functions and objects with perimeters.

OPCODE 17

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *color* specifies the new color to define for line-drawing.

BINDING

```
contrl[0] = 17;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;  
  
intin[0] = color;  
  
vdi();  
  
return intout[0];
```

RETURN VALUE This function returns the new color set (within bounds).

SEE ALSO `vst_color()`, `vsm_color()`, `vsf_color()`

vsl_ends()

VOID `vsl_ends(handle, start, end)`

WORD *handle*, *start*, *end*;

`vsl_ends()` sets the style of end point for the starting and ending points of lines drawn by the **VDI** in line-drawing functions and perimeter drawing.

OPCODE 108

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *start* and *end* specify the type of end cap to use at the start and end of lines respectively as follows:

Name	<i>start/end</i>	Shape
SQUARE	0	
ARROWED	1	
ROUND	2	

BINDING

```
contrl[0] = 108;  
contrl[1] = 0;  
contrl[3] = 2;  
contrl[6] = handle;
```

```

intin[0] = start;
intin[1] = end;

vdi();

```

SEE ALSO `vsl_type()`

vsl_type()

WORD `vsl_type(handle, type)`

WORD `handle, type;`

`vsl_type()` defines the style of line used in line-drawing functions and perimeter drawing.

OPCODE 15

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *type* defines the style of line as follows:

Name	type	Style
SOLID	0	
LDASHED	1	
DOTTED	2	
DASHDOT	3	
DASH	4	
DASHDOTDOT	5	
USERLINE	6	User-defined with <code>vsl_udsty()</code> .

BINDING

```

contrl[0] = 15;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

```

```
    intin[0] = type;
    vdi();
    return intout[0];
```

RETURN VALUE **vsl_style()** returns the newly set line type.

SEE ALSO **vsl_udsty()**

vsl_udsty()

VOID **vsl_udsty(*handle*, *pattern*)**

WORD *handle*, *pattern*;

vsl_udsty() sets the user-defined line type.

OPCODE 113

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *pattern* is a **WORD** which defines the **USERLINE** style. It is essentially a bit mask which is applied to a solid line and repeated along the length of the line. A value of 0xFFFF would create a solid line. A value of 0xAAAA would produce a line where every other pixel was set.

BINDING

```
    contrl[0] = 113;
    contrl[1] = 0;
    contrl[3] = 1;
    contrl[6] = handle;

    intin[0] = pattern;

    vdi();
```

COMMENTS You must call **vsl_style(*handle*, 6)** to actually utilize this style.

SEE ALSO **vsl_style()**

vsl_width()

VOID vsl_width(*handle*, *width*)

WORD *handle*, *width*;

vsl_width() determines the width of lines drawn with line-drawing functions and as perimeters to other objects.

OPCODE 16

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *width* specifies the width future lines drawn will be.

BINDING

```
contrl[0] = 16;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;  
  
intin[0] = width;  
  
vdi();
```

COMMENTS The **VDI** is only capable of drawing lines an odd number of pixels thick. Values will be rounded down to the first odd number.

Setting a line width higher than 1 may nullify line styles other than solid. Check **vq_extnd()** for details.

SEE ALSO vq_extnd()

vsm_choice()

WORD vsm_choice(*handle*, *xout*)

WORD *handle*;

WORD **xout*;

vsm_choice() returns the current value of the ‘choice’ device.

OPCODE 30

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>xout</i> points to a WORD which is filled in on function exit with the current value of the choice device.
BINDING	<pre>contrl[0] = 30; contrl[1] = contrl[3] = 0; contrl[6] = handle; vdi(); *xout = intout[0]; return contrl[4];</pre>
RETURN VALUE	vsm_choice() returns 1 if an input from the ‘choice’ device was made or 0 otherwise.
SEE ALSO	vsin_mode() , vrq_choice()

vsm_color()

WORD **vsm_color**(*handle*, *color*)

WORD *handle*, *color*;

vsm_color() defines the color used to render markers.

OPCODE	20
AVAILABILITY	Supported by all drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>color</i> specifies the new color to define for markers.
BINDING	<pre>contrl[0] = 20; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; vdi(); return intout[0];</pre>
RETURN VALUE	vsm_color() returns the new marker color actually set (within bounds).
SEE ALSO	v_pmarker() , vsl_color() , vst_color() , vsf_color()

vsm_height()

WORD `vsm_height(handle, size)`

WORD `handle, size;`

`vsm_height()` sets the height of markers.

OPCODE 19

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *size* specifies the height (and width) of markers to draw in pixels.

BINDING

```

contrl[0] = 19;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = size;

vdi();

return intout[0];

```

RETURN VALUE `vsm_height()` returns the marker height actually set.

COMMENTS The **DOT** marker is not affected by this call. It is always one pixel high and wide.

SEE ALSO `v_pmarker()`

vsm_locator()

WORD `vsm_locator(handle, mx, my, xout, yout, term)`

WORD `handle, mx, my;`

WORD `*xout, *yout, *term;`

`vsm_locator()` receives data from the 'locator' device in sample mode.

OPCODE 28

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS *handle* specifies a valid workstation handle. The mouse pointer is initially drawn

at location (*mx*, *my*). The call returns with the final position of the mouse in the **WORD**s pointed to by *xout* and *yout*.

The **WORD** pointed to by *term* will be filled in with a value which specifies the ASCII value of the key pressed. *term* will be set to 0x20 if the left mouse button was pressed or 0x21 if the right mouse button was pressed.

BINDING

```
contrl[0] = 28;  
contrl[1] = 1;  
contrl[3] = 0;  
contrl[6] = handle;  
  
ptsin[0] = mx;  
ptsin[1] = my;  
  
vdi();  
  
*xout = ptsout[0];  
*yout = ptsout[1];  
  
*term = intout[0];  
  
return ((contrl[4] << 1) | contrl[2]);
```

RETURN VALUE

vsm_locator() returns one of the following based on its result:

Return Value	Meaning
0	Mouse has not moved nor was any key pressed.
1	Mouse has been moved (<i>xout</i> and <i>yout</i> are valid).
2	Key or mouse button has been struck (<i>term</i> is valid).
3	Mouse has moved and a key or mouse button has been struck (<i>xout</i> , <i>yout</i> , and <i>term</i> are valid).

CAVEATS

Using this call will confuse the **AES**.

SEE ALSO

vrq_locator(), **vsin_mode()**

vsm_string()

WORD **vsm_string()** (*handle*, *maxlen*, *echo*, *echoxy*, *str*)

WORD *handle*, *maxlen*, *echo*;

WORD **echoxy*;

char **str*;

vsm_string() retrieves input from the 'string' device.

OPCODE

31

AVAILABILITY	This call is not guaranteed to be available with any driver and its use should therefore be restricted.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. This call inputs characters from the keyboard into the buffer pointed to by <i>str</i> up to (<i>maxlen</i> + 1) characters. If <i>echo</i> is set to 1, characters are echoed to the screen at the location given by the two WORD s pointed to by <i>outxy</i> . If echo is set to 0, no echoing is performed.
BINDING	<pre>WORD i; contrl[0] = 31; contrl[1] = 1; contrl[3] = 2; contrl[6] = handle; intin[0] = maxlen; intin[1] = echo; ptsin[0] = echoxy[0]; ptsin[1] = echoxy[1]; vdi(); for(i = 0; i < contrl[4]; i++) str[i] = (char)intout[i]; return contrl[4];</pre>
RETURN VALUE	vsm_string() returns the number of characters actually read.
CAVEATS	Using this function will confuse the AES .
COMMENTS	Though this binding does not allow for it, if <i>maxlen</i> is specified as negative, then as many as ($ maxlen + 1$) characters will be read as keycodes rather than ASCII codes. The values in <i>intout</i> will occupy the full WORD rather than just the lower eight bits. A custom binding could be used to take advantage of this.
SEE ALSO	vsin_mode()

vsm_type()

WORD vsm_type(*handle*, *type*)

WORD *handle*, *type*;

vsm_type() sets the current type of marker.

OPCODE 18

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *type* changes the marker type as follows:

Name	<i>type</i>	Shape
MRKR_DOT	1	Single Pixel
MRKR_PLUS	2	
MRKR_ASTERISK	3	
MRKR_BOX	4	
MRKR_CROSS	5	
MRKR_DIAMOND	6	
—	7...	Device Dependent

BINDING

```

contrl[0] = 18;
contrl[1] = 0;
contrl[3] = type;
contrl[6] = handle;

intin[0] = type;

vdi();

```

RETURN VALUE `vsm_type()` returns the type of marker actually set.

SEE ALSO `v_pmarker()`

vsm_valuator()

VOID `vsm_valuator(handle, x, xout, term, status)`

WORD `handle, x;`

WORD `*xout, *term, *status;`

`vsm_valuator()` retrieves input from the ‘valuator’ device in sample mode.

OPCODE 29

AVAILABILITY This call is not guaranteed to be available with any driver and its use should therefore be restricted.

PARAMETERS *handle* specifies a valid workstation handle. *x* sets the initial value of the ‘valuator’. The **WORD** pointed to by *xout* is filled in with the final value of the device. If a key was pressed its ASCII code is returned in the **WORD** pointed to by *term*. The **WORD** pointed to by *status* contains a value as follows:

<i>status</i>	Meaning
0	No input was taken.
1	Valuator changed.
2	Key press occurred.

```
BINDING
    contrl[0] = 29;
    contrl[1] = 0;
    contrl[3] = 1;
    contrl[6] = handle;

    intin[0] = x;

    vdi();

    *xout = intout[0];
    *term = intout[1];

    *status = contrl[4];
```

SEE ALSO `vsin_mode(), vrq_valuator()`

vsp_message()

VOID vsp_message(*handle*)

WORD *handle*;

vsp_message() causes the suppression of palette driver messages from the screen.

OPCODE	5
SUB-OPCODE	95
AVAILABILITY	Supported by all camera drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle.
BINDING	<pre>contrl[0] = 5; contrl[1] = contrl[3] = 0; contrl[5] = 95; contrl[6] = handle; vdi();</pre>
SEE ALSO	vqp_error()

vsp_save()

VOID vsp_save(*handle*)

WORD *handle*;

vsp_save() saves the current state of the driver to disk.

OPCODE	5
SUB-OPCODE	94
AVAILABILITY	Supported by all camera drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle.
BINDING	<pre>contrl[0] = 5; contrl[1] = contrl[3] = 0; contrl[5] = 94; contrl[6] = handle; vdi();</pre>

vsp_state()

VOID vsp_state(*handle*, *port*, *film*, *lightness*, *interlace*, *planes*, *indexes*)

WORD *handle*, *port*, *film*, *lightness*, *interlace*, *planes*;

WORD **indexes*;

vsp_state() sets the palette driver state.

OPCODE 5

SUB-OPCODE 93

AVAILABILITY Supported by all camera drivers.

PARAMETERS *handle* specifies a valid workstation handle. *port* specifies the communication port number of the camera device. *film* specifies the index of the desired type of film (0–4).

lightness specifies the modification to apply to the camera's default f-stop setting (-3–3). A value of 0 uses the default setting. A value of -3 results in an exposure of half of the default length whereas a value of 3 doubles the exposure time. *interlace* is set to 0 for non-interlaced or 1 for interlaced output.

planes specifies the number of planes to output (1–4). *indexes* points to an array of 16 **WORD**s which define the color codes for the palette.

BINDING

WORD *i*;

```
contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 20;
contrl[5] = 93;
contrl[6] = handle;
```

```
intin[0] = port;
intin[1] = film;
intin[2] = lightness;
intin[3] = interlace;
intin[4] = planes;
for(i = 0; i < 16; i++)
    intin[i + 5] = indexes[i];
```

```
vdi();
```

SEE ALSO **vqp_state()**

vst_alignment()

VOID vst_alignment(*handle*, *halign*, *valign*, **hout*, **vout*)

WORD *handle*, *halign*, *valign*;

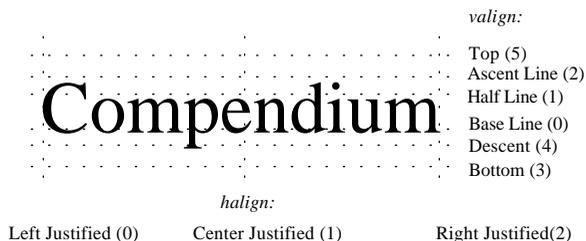
WORD **hout*, **vout*;

vst_alignment() affects the vertical and horizontal alignment of normal and justified text.

OPCODE 39

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *halign* and *valign* affects where the coordinate specified by **v_gtext()** or **v_justified()** actually applies to as follows:



On return, the **WORD**s pointed to by *hout* and *vout* are filled in with the values actually set.

BINDING

```
contrl[0] = 39;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = halign;
intin[1] = valign;

vdi();

*hout = intout[0];
*vout = intout[1];
```

SEE ALSO **v_gtext()**, **v_justified()**

vst_arbpt()

WORD vst_arbpt(*handle*, *point*, *wchar*, *hchar*, *wcell*, *hcell*)

WORD *handle*;

WORD *point*;

WORD **wchar*, **hchar*, **wcell*, **hcell*;

vst_arbpt() selects any point size for an outline font.

OPCODE 246

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *point* specifies the point size at which to render outline text.

Upon return, the **WORDS** pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING

```
contrl[0] = 246;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;
```

```
intin[0] = point;
```

```
vdi();
```

```
*wchar = ptsout[0];  
*hchar = ptsout[1];  
*wcell = ptsout[2];  
*hcell = ptsout[3];
```

```
return intout[0];
```

RETURN VALUE **vst_arbpt()** returns the point size actually selected.

COMMENTS This call only works with outline fonts, however, it is not restricted by the point sizes listed in the 'ASSIGN.SYS' file.

To specify a fractional point size, use **vst_arbpt32()**.

SEE ALSO **vst_arbpt32()**, **vst_point()**, **vst_height()**

vst_arbpt32()

fix31 vst_arbpt(*handle*, *point*, *wchar*, *hchar*, *wcell*, *hcell*)

WORD *handle*;

fix31 *point*;

WORD **wchar*, **hchar*, **wcell*, **hcell*;

vst_arbpt32() selects a fractional point size for an outline font.

OPCODE 246

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *point* specifies the point size at which to render outline text as a **fix31** value.

Upon return, the **WORDS** pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING

```
contrl[0] = 246;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = (WORD)(point >> 16);
intin[1] = (WORD)(point & 0xFFFF);

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return (((fix31)intout[0] << 16) | (fix31)intout[1]);
```

RETURN VALUE **vst_arbpt32()** returns the point size actually selected.

COMMENTS This call only works with outline fonts, however, it is not restricted by the point sizes listed in the 'ASSIGN.SYS' file.

SEE ALSO **vst_arbpt()**, **vst_point()**, **vst_height()**

vst_charmap()

VOID vst_charmap(*handle*, *mode*)

WORD *handle*, *mode*;

vst_charmap() chooses between the standard Atari ASCII interpretation of text strings or translation of Bitstream character indexes.

OPCODE 236

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *mode* should be **MAP_ATARI** (1) to specify Atari ASCII characters or **MAP_BITSTREAM** (0) for Bitstream mappings.

BINDING

```

contrl[0] = 236;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = mode;

vdi();

```

COMMENTS Bitstream character indexes are **WORD** sized rather than **BYTE** sized. A list of Bitstream character mappings can be found in Appendix G.

vst_color()

WORD vst_color(*handle*, *color*)

WORD *handle*, *color*;

vst_color() sets the current text color.

OPCODE 22

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *color* specifies the new color to apply to text.

BINDING

```

contrl[0] = 22;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

```

```
intin[0] = color;  
vdi();  
return intout[0];
```

RETURN VALUE `vst_color()` returns the text color actually set (within bounds).

SEE ALSO `vsl_color()`, `vsm_color()`, `vsf_color()`

vst_effects()

WORD `vst_effects(handle, effects)`

WORD `handle, effects;`

`vst_effects()` defines which special effects are to be applied to text.

OPCODE 106

AVAILABILITY Supported by all drivers.

PARAMETERS `handle` specifies a valid workstation handle. `effects` is a bit mask which specifies one or more special effects to apply to text as follows:

Name	Bit	Meaning
THICKENED	0	Thickened
LIGHT	1	Lightened
SKEWED	2	Skewed
UNDERLINED	3	Underlined
OUTLINED	4	Outlined
SHADOWED	5	Shadowed (not currently supported)

BINDING

```
contrl[0] = 106;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;  
  
intin[0] = effects;  
  
vdi();  
  
return intout[0];
```

RETURN VALUE `vst_effects()` returns the actual effects set by the call.

COMMENTS Special effects do not, in general, work well with outline text (besides

underlining). To compensate, most type families have bold and italic faces in addition to the `vst_skew()` call.

SEE ALSO `vst_skew()`

vst_error()

VOID `vst_error(handle, mode, error)`

WORD `handle, mode;`

WORD `*error;`

`vst_error()` provides a method to obtain errors from **GDOS** and suppress text messages on screen.

OPCODE 245

AVAILABILITY Available only with **FONTGDOS**, **FSM**, or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *mode* specifies the error reporting mode. A value of **SCREEN_ERROR** (1) (default) causes error messages to be outputted to the screen as text.

A value of **APP_ERROR** (0) suppresses these messages and instead places an error code in the **WORD** pointed to by *error* whenever an error occurs leaving it up to the application to process errors correctly. Prior to making this call and after each reported error, the application is responsible for resetting the value pointed to by *error* to 0. The following is a list of possible error codes:

Name	<i>error</i>	Meaning
NO_ERROR	0	No error.
CHAR_NOT_FOUND	1	Character not found in font.
FILE_READERR	8	Error reading file.
FILE_OPENERR	9	Error opening file.
BAD_FORMAT	10	Bad file format.
CACHE_FULL	11	Out of memory/cache full.
MISC_ERROR	-1	Miscellaneous error.

BINDING

```

contrl[0] = 245;
contrl[1] = 0;
contrl[3] = 3;
contrl[6] = handle;

intin[0] = mode;
*(LONG *)&intin[1] = (LONG)error;

```

```
vdi();
```

COMMENTS Once setting the error mode to 0, an application should check the error variable after each of the following calls:

<code>v_gtext()</code>	<code>v_justified()</code>	<code>vst_point()</code>
<code>vst_height()</code>	<code>vst_font()</code>	<code>vst_arbpt()</code>
<code>vqt_advance()</code>	<code>vst_setsize()</code>	<code>vqt_fontinfo()</code>
<code>vqt_name()</code>	<code>vqt_width()</code>	<code>vqt_extnt()</code>
<code>v_opnwk()</code>	<code>v_opnvwk()</code>	<code>vst_load_fonts()</code>
<code>vst_unload_fonts()</code>	<code>v_ftext()</code>	<code>vqt_f_extnt()</code>

vst_font()

WORD `vst_font(handle, index)`

WORD `handle, index;`

`vst_font()` sets the current text font.

OPCODE 21

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *index* specifies the index (as returned by `vqt_name()`) of the font to enable.

BINDING

```
contrl[0] = 21;  
contrl[1] = 0;  
contrl[3] = 1;  
contrl[6] = handle;
```

```
intin[0] = index;
```

```
vdi();
```

```
return intout[0];
```

RETURN VALUE `vst_font()` returns the index of the font actually set.

SEE ALSO `vqt_name()`

vst_height()

VOID vst_height(*handle*, *height*, *wchar*, *hchar*, *wcell*, *hcell*)

WORD *handle*, *height*;

WORD **wchar*, **hchar*, **wcell*, **hcell*;

vst_height() sets the height of the current text face (in pixels).

OPCODE 12

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *height* specifies the height (in pixels) at which to render text. Upon return, the **WORD**s pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING

```

contrl[0] = 12;
contrl[1] = 1;
contrl[3] = 0;
contrl[6] = handle;

ptsin[0] = 0;
ptsin[1] = height;    /* Passed in ptsin[1] because of VDI bug.
                       */

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

```

COMMENTS **vst_height()** works on both bitmap and outline fonts. The font will be scaled to fit within the height given. This doesn't always give good results with bitmap text.

SEE ALSO **vst_point()**, **vst_arbpt()**

vst_kern()

VOID vst_kern(*handle*, *tmode*, *pmode*, *tracks*, *pairs*)

WORD *handle*, *tmode*, *pmode*;

WORD **tracks*, **pairs*;

vst_kern() sets the track and pair kerning values.

OPCODE 237

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *tmode* specifies the track kerning mode as follows:

Name	<i>tmode</i>	Meaning
TRACK_NONE	0	No track kerning
TRACK_NORMAL	1	Normal track kerning
TRACK_TIGHT	2	Tight track kerning
TRACK_VERYTIGHT	3	Very tight track kerning

Setting *pmode* to **PAIR_ON** (1) turns pair kerning on. Setting it to **PAIR_OFF** (0) turns pair kerning off.

The **WORD** pointed to by *tracks* is filled in with the track kerning mode actually set. *pairs* points to a **WORD** which is filled in with the number of defined character kerning pairs.

BINDING

```
contrl[0] = 237;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = tmode;
intin[1] = pmode;

vdi();

*tracks = intout[0];
*pairs = intout[1];
```

SEE ALSO **vqt_trackkern()**, **vqt_pairkern()**

vst_load_fonts()

WORD **vst_load_fonts(*handle*, *rsrvd*)**

WORD *handle*, *rsrvd*;

vst_load_fonts() loads disk-based font information into memory.

OPCODE 119

AVAILABILITY Available with any form of **GDOS**.

PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>rsrvd</i> is currently unused and must be 0.
BINDING	<pre> contrl[0] = 119; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; intin[0] = rsrvd; vdi(); </pre>
RETURN VALUE	vst_load_fonts() returns the number of extra fonts loaded.
COMMENTS	Calling this function more than once before calling vst_unload_fonts() will return 0.
SEE ALSO	vst_unload_fonts() , vqt_name()

vst_point()

WORD vst_point(*handle*, *point*, *wchar*, *hchar*, *wcell*, *hcell*)

WORD *handle*, *height*;

WORD **wchar*, **hchar*, **wcell*, **hcell*;

vst_point() sets the height of the current text face in points (1/72 inch).

OPCODE	107
AVAILABILITY	Supported by all drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>point</i> specifies a valid point size to set the current text face to. This means an appropriate bitmap font or a point size enumerated in the 'EXTEND.SYS' file.

Upon return, the **WORD**s pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING	<pre> contrl[0] = 107; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; intin[0] = point; vdi(); </pre>
----------------	--

```
*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return intout[0];
```

RETURN VALUE `vst_point()` returns the point size actually set.

COMMENTS If a point size which doesn't exist for the current face is selected, the next valid size down is selected.

SEE ALSO `vst_arbpt()`, `vst_height()`

`vst_rotation()`

WORD `vst_rotation(handle, angle)`

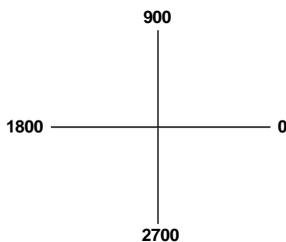
WORD *handle, angle*;

`vst_rotation()` sets the angle at which graphic text is drawn.

OPCODE 13

AVAILABILITY Supported by all drivers. For specific character rotation abilities, check the values returned in `vq_extnd()`.

PARAMETERS *handle* specifies a valid workstation handle. *angle* specifies the angle at which to rotate text in tenths of degrees as follows:



BINDING

```
contrl[0] = 13;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;
```

```
intin[0] = angle;
```

```
vdi();
```

```
return intout[0];
```

RETURN VALUE `vst_rotation()` returns the value of rotation actually set.

COMMENTS Bitmap fonts may only be rotated at 0, 90, and 270 degrees. Outline fonts may be rotated at any angle with **FSM**.

vst_scratch()

VOID `vst_scratch(handle, mode)`

WORD `handle, mode;`

`vst_scratch()` allows **FSMGDOS** or **SpeedoGDOS** to change its method of allocating a scratch buffer for better efficiency.

OPCODE 244

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *mode* specifies the scratch buffer allocation mode as follows:

Name	mode	Meaning
SCRATCH_BOTH	0	Scratch buffers should be allocated which are large enough for FSM/Speedo and bitmap fonts with any combination of special effects.
SCRATCH_BITMAP	1	Scratch buffers should be allocated which are large enough for FSM/Speedo fonts with no effects and bitmap fonts with effects.
SCRATCH_NONE	2	Scratch buffers should be allocated which are large enough for FSM/Speedo fonts and bitmap fonts with no special effects.

BINDING

```

contrl[0] = 244;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = mode;

vdi();

```

COMMENTS Atari recommends that at least mode 1 be set prior to a `vst_load_fonts()` call to prevent scratch buffer overruns.

The size of the scratch buffer is based on the size of the largest point size specified in the 'EXTEND.SYS' file. Attempting to add effects to a character higher in point size than this will cause a buffer overrun.

vst_setsize()

WORD vst_setsize(*handle*, *point*, *wchar*, *hchar*, *wcell*, *hcell*)

WORD *handle*;

WORD *point*;

WORD **wchar*, **hchar*, **wcell*, **hcell*;

vst_setsize() sets the width of outline characters.

OPCODE 252

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle.

point specifies the width of the character in points (1/72 inch). A value for *point* equivalent to the same point size specified in **vst_arbpt()** will result in a correctly proportioned character.

Upon return, the **WORDS** pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

```
BINDING      contrl[0] = 252;
               contrl[1] = 0;
               contrl[3] = 1;
               contrl[6] = handle;

               intin[0] = point;

               vdi();

               *wchar = ptsout[0];
               *hchar = ptsout[1];
               *wcell = ptsout[2];
               *hcell = ptsout[3];

               return intout[0];
```

RETURN VALUE **vst_setsize()** returns the size actually set.

COMMENTS This call only works with outline fonts. At the next **vst_point()**, **vst_height()**, or **vst_arbpt()** the size will be reset to the correct proportions (width in points = height in points).

To set a fractional size, use **vst_setsize32()**.

SEE ALSO `vst_arbpt()`, `vst_setsize32()`

vst_setsize32()

fix31 `vst_setsize(handle, point, wchar, hchar, wcell, hcell)`

WORD *handle*;

fix31 *point*;

WORD **wchar, *hchar, *wcell, *hcell*;

`vst_setsize()` sets the width of outline characters as a **fix31** fractional value.

OPCODE 252

AVAILABILITY Available only with **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle.

point specifies the width of the character in points (1/72 inch). A value for *point* equivalent to the same point size specified in `vst_arbpt()` will result in a correctly proportioned character.

Upon return, the **WORD**s pointed to by *wchar*, *hchar*, *wcell*, and *hcell* will be filled in with the width and height of the character and the width and height of the character cell respectively.

BINDING

```

contrl[0] = 252;
contrl[1] = 0;
contrl[3] = 2;
contrl[6] = handle;

intin[0] = (WORD)(point >> 8);
intin[1] = (WORD)point;

vdi();

*wchar = ptsout[0];
*hchar = ptsout[1];
*wcell = ptsout[2];
*hcell = ptsout[3];

return ((fix31)intout[0] << 16) | (fix31)intout[1];

```

RETURN VALUE `vst_setsize32()` returns the size actually set.

COMMENTS This call only works with outline fonts. At the next `vst_point()`, `vst_height()`, or `vst_arbpt()` the size will be reset to the correct proportions (width in points = height in points).

SEE ALSO `vst_setsize()`, `vst_arbpt()`

vst_skew()

WORD `vst_skew(handle, skew)`

WORD `handle, skew;`

`vst_skew()` sets the skew amount for fonts.

OPCODE 253

AVAILABILITY Available only with **FSMGDOS** or **SpeedoGDOS**.

PARAMETERS *handle* specifies a valid workstation handle. *skew* specifies the amount to skew in tenths of degrees from -900 to 900. Negative values skew to the left and positive values skew to the right. *skew* values of -900 or 900 will result in a flat line.

BINDING

```
contrl[0] = 253;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

intin[0] = skew;

vdi();

return intout[0];
```

RETURN VALUE `vst_skew()` returns the skew value actually set.

COMMENTS This call should only be used with outline fonts. Note that this call generates a true ‘skew’ effect independent of that generated by `vst_effects()` which is an algorithmic ‘skew’. The algorithmic ‘skew’ may be used on bitmap fonts but is rather unpleasant applied to outline fonts.

SEE ALSO `vst_effects()`

vst_unload_fonts()

VOID `vst_unload_fonts(handle, select)`

WORD `handle, select;`

`vst_unload_fonts()` frees memory associated with disk-loaded fonts.

OPCODE 120

AVAILABILITY	Available under any form of GDOS .
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>select</i> is reserved and should be 0.
BINDING	<pre> contrl[0] = 120; contrl[1] = 0; contrl[3] = 1; contrl[6] = handle; intin[0] = select; vdi(); </pre>
SEE ALSO	vst_load_fonts()

vswr_mode()

WORD vswr_mode(*handle*, *mode*)

WORD *handle*, *mode*;

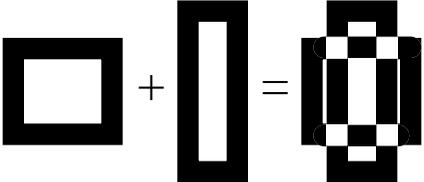
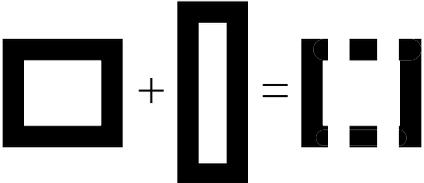
vswr_mode() defines the writing mode for rendering **VDI** objects.

OPCODE 32

AVAILABILITY Supported by all drivers.

PARAMETERS *handle* specifies a valid workstation handle. *mode* specifies a writing mode as follows:

Name	mode	Example
MD_REPLACE	1	
MD_TRANS	2	

MD_XOR	3	
MD_ERASE	4	

BINDING

```

contrl[0] = 32;
contrl[1] = 0;
contrl[3] = 1;
contrl[6] = handle;

```

```

intin[0] = mode;

```

```

vdi();

```

```

return intout[0];

```

RETURN VALUE

vswr_mode() returns the writing mode set.

COMMENTS

In true-color modes, **MD_ERASE** and **MD_TRANS** work a little differently, they write (or avoid writing on) whatever color is currently held in VDI color 0 (as opposed to the actual register reference of 0).

vt_alignment()

VOID **vt_alignment()** (*handle*, *dx*, *dy*)

WORD *handle*, *dx*, *dy*;

vt_alignment() allows an offset to be specified that will be applied to all coordinates output from the graphics tablet.

OPCODE

5

SUB-OPCODE

85

AVAILABILITY

Supported by all tablet drivers.

PARAMETERS

handle specifies a valid workstation handle. *dx* and *dy* are the delta offsets from

(0, 0) to apply to values from the graphics tablet.

BINDING

```

contrl[0] = 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 85;
contrl[6] = handle;

intin[0] = dx;
intin[1] = dy;

vdi();

```

COMMENTS This call is used to ‘fine-tune’ the true starting point of the tablet.

SEE ALSO vt_origin()

vt_axis()

VOID vt_axis(*handle*, *xres*, *yres*, **xout*, **yout*)

WORD *handle*, *xres*, *yres*;

WORD **xout*, **yout*;

vt_axis() sets the horizontal and vertical resolution for the graphics tablet (in lines).

OPCODE 5

SUB-OPCODE 82

AVAILABILITY Supported by all tablet drivers.

PARAMETERS *handle* specifies a valid workstation handle. *xres* and *yres* specify the new horizontal and vertical resolution of the tablet respectively. Upon return, the **WORD**s pointer to by *xout* and *yout* are filled in with the resolution actually set.

BINDING

```

contrl[0]= 5;
contrl[1] = 0;
contrl[3] = 2;
contrl[5] = 82;
contrl[6] = handle;

intin[0] = xres;
intin[1] = yres;

vdi();

*xout = intout[0];
*yout = intout[1];

```

SEE ALSO `vt_alignment()`, `vt_origin()`

vt_origin()

VOID `vt_origin(handle, xorigin, yorigin)`

WORD `handle, xorigin, yorigin;`

`vt_origin()` sets the origin point for the tablets' upper-left point.

OPCODE 5

SUB-OPCODE 83

AVAILABILITY Supported by all tablet drivers.

PARAMETERS *handle* specifies a valid workstation handle. *xorigin* and *yorigin* specify the new upper-left point recognized by the tablet.

BINDING

```
contrl[0] = 5;  
contrl[1] = 0;  
contrl[3] = 2;  
contrl[5] = 83;  
contrl[6] = handle;  
  
intin[0] = xorigin;  
intin[1] = yorigin;  
  
vdi();
```

SEE ALSO `vt_axis()`, `vt_alignment()`

vt_resolution()

VOID `vt_resolution(handle, xres, yres, *xout, *yout)`

WORD `xres, yres;`

WORD `*xout, *yout;`

`vt_resolution()` sets the horizontal and vertical resolution of the graphics tablet (in lines per inch).

OPCODE 5

SUB-OPCODE 81

AVAILABILITY	Supported by all tablet drivers.
PARAMETERS	<i>handle</i> specifies a valid workstation handle. <i>xres</i> and <i>yres</i> specify the new horizontal and vertical resolution values for the tablet respectively. Upon return, the WORDS s pointed to by <i>xout</i> and <i>yout</i> are filled in with the values actually set.
BINDING	<pre>contrl[0] = 5; contrl[1] = 0; contrl[3] = 2; contrl[5] = 81; contrl[6] = handle; intin[0] = xres; intin[1] = yres; vdi(); *xout = intout[0]; *yout = intout[1];</pre>
SEE ALSO	vt_axis()