PIXFUN PIXFUN

SYNTAX

ASCII	PIXFUN mode
FORTRAN Call	CALL PIXFUN (MODE)
Binary	[59] [mode] (2 bytes)
	59 decimal = 073 octal = 3B hex

FUNCTION

The PIXFUN command sets the pixel processor mode. All operations which affect the image memory (with the exception of FLOOD) are performed by the pixel processor. These include graphics primitives, pixel mover, and image transfer operations.

The operation to be performed by the pixel processor is specified by mode. A character string can be substituted for the mode number when entered from the local alphanumeric terminal.

The valid pixel processor modes are:

Mode	ASCII Mnemonic	Operation
0	INS	Directly insert new data. (Default)
1	SUBI	Subtract image data from new data.
2	SUBN	Subtract new data from image data.
3	ADD	Add new data to image data.
4	XOR	XOR new data to image data.
5	OR	OR new data to image data.
6	AND	AND new data to image data.
7	PRESET	Write all ones into image memory.
8	CONDITIONAL	Insert new data, but inhibit writing of all pixels whose value is 0,0,0.

ASCII PARAMETER

mode	The mode may be 0 to 8; the mode can also be given as a character string if the data is sent in ASCII mode.

PIXFUN PIXFUN

FORTRAN PARAMETER

INTEGER*2 MODE

EXAMPLE

```
! LUT8 1 0,0,255          ; Set the color out for LUT index 1 to blue.
! LUT8 2 255,0,0          ; Set the color out for LUT index 2 to red.
! PRMFIL ON               ; Select filled primitives.
! VAL8 1                  ; Set current pixel value to 1 (blue).
! FLOOD                   ; Flood displayed image; screen turns blue.
! VAL8 2                  ; Set current pixel value to 2 (red).
! PIXFUN INS              ; Set pixel processor mode to INS (default).
! CIRCLE 50               ; Draw a filled red circle.
! MOVABS 100 100          ; Move current point to 100,100.
! PIXFUN XOR              ; Set pixel processor mode to XOR.
! CIRCLE 50               ; Draw a filled magenta circle (the result when blue
                            is XORed with red).
! MOVABS -100 100         ; Move current point to -100,100.
! PIXFUN AND              ; Set pixel processor mode to AND.
! CIRCLE 50               ; Draw a filled black circle (the result when blue
                            is ANDed with black).
```

PIXMOV PIXMOV

SYNTAX

ASCII	PIXMOV
FORTRAN Call	CALL PIXMOV
Binary	[187] (1 byte)
	187 decimal = 273 octal = BB hex

FUNCTION

The PIXMOV command performs a block pixel move. This command moves pixel data from a rectangular source window to a destination window of the same size.

The source window is specified by CREG 11 and CREG 12, which contain diagonally opposite corners of the window. The destination window is specified by CREG 13 and CREG 14. The pixel at the location in CREG 11 (the source window corner) is transferred to the location in CREG 13 (the destination window corner). CREG 14 controls the direction of scanning of pixels into the destination window. The direction of the displacement of CREG 14 relative to CREG 13 defines the position of the diagonal corner of the destination window, allowing mirroring.

EXAMPLE

```
! MOVABS 0 0              ; Move current point to 0,0.
! DRWABS 30 30            ; Draw a triangle.
! DRWABS 45 0
! DRWABS 0 0
! CLOAD 11 -50 50         ; Define source window in center of screen.
! CLOAD 12 50 -50
! CLOAD 13 -256 255       ; Define destination window in upper left corner
                            of screen.
! CLOAD 14 -156 155
! PIXMOV                  ; Move triangle in center window to window in
                            upper left corner of screen.
! CLOAD 13 -156 155       ; Redefine destination window. Specify corners
                            in opposite order.
! CLOAD 14 -256 255
! PIXMOV                  ; Move triangle in center window to window in
                            upper left corner of screen. Triangle is
                            mirrored about both the x and y axes.
```

PMCTL PMCTL

SYNTAX

ASCII	PMCTL flag srcmem destmem reserved
FORTRAN Call	CALL PMCTL (SF, CF1, CF2, MF, REDRTE, GRERTE, BLURTE)
Binary	[191] [0] [SRCMEM] [DESTMEM] [0] [0] [0] [0]
	(8 bytes)
	191 decimal = 277 octal = BF hex

FUNCTION

The PMCTL command sets up the routing for the pixel mover, indicating how pixels are moved between memory units.

The flag parameter is currently reserved and ignored by the Model One/80. Similarly, the fourth parameter (which is 4 bytes) is reserved. You should set the flag and reserved parameters to zero.

The srcmem and destmem parameters identify memory units (as in MEMSEL), not memory planes.

Selection of a non-existant memory unit does not cause an error. The RDCNFG command can be used to show which memory units have been configured.

The Model One/80 PMCTL command is incompatible with the Model One/25 PMTCL command.

ASCII PARAMETERS

flag	Eight-bit parameter reserved for future use; currently it must be set to zero.
srcmem	Eight-bit parameter specifying the source memory unit for the pixel move. The range is 0 to 15.
destmem	Eight-bit parameter specifying the destination memory unit for the pixel move. The range is 0 to 15.
reserved	Thirty-two bit parameter reserved for future use; currently it should be set to zeros.

PMCTL PMCTL

FORTRAN PARAMETERS

INTEGER*2 SF, CF1, CF2, MF, REDRTE, CRERTE, BLURTE

SF, MF, REDRTE, GREGRTE, and BLURTE should all be set to zero.

POINT POINT

SYNTAX

ASCII	POINT
FORTRAN Call	CALL POINT
Binary	[136] (1 byte)
	136 decimal = 210 octal = 88 hex

FUNCTION

The POINT command sets the current point (CREG 0) to the current pixel value (VREG 0). The current point and the current pixel value remain unchanged.

EXAMPLE

```
! LUT8 5 255,0,0          ; Set the color out for LUT index 5 to red.
! LUT8 6 255,255,255      ; Set the color out for LUT index 6 to white.
! MOVABS 0 0              ; Move current point to 0,0.
! PRMFIL ON               ; Select filled primitives.
! VAL8 5                  ; Change current pixel value to 5 (red).
! CIRCLE 30               ; Draw a red circle.
! VAL8 6                  ; Change current pixel value to 6 (white).
! POINT                   ; Set pixel at 0,0 to 6 (white). You can see a
                            white point in the center of the red circle.
```

POKE POKE

SYNTAX

ASCII	POKE addr, data
FORTRAN Call	CALL POKE (IADDR, IDATA)
Binary	[190] [highaddr][lowaddr] [highdata][lowdata] (5 bytes)

190 decimal = 276 octal = BE hex

FUNCTION

The POKE command writes a given word of data into a given address, addr, in central processor memory. The POKE command is intended to be used in conjunction with the PEEK command as a debugging tool for implementors, and not as a general purpose command for users.

ROM memory runs from 0 to 7FFF hex. POKEs into this memory have no effect.

RAM memory runs from 8000 to FFFF hex. You should not POKE into the memory area from 8000 to BFFF, which is the base RAM. POKEs into this area can crash the central processor.

ASCII PARAMETERS

addr	16-bit integer specifying the address in central processor memory in which to place data.
data	16-bit integer data.

FORTRAN PARAMETERS

INTEGER*2 IADDR, IDATA

EXAMPLE

```
! PEEK #C000              ; Display contents of memory location C000 hex.
  02DC                      (Possible response from Model One.)
! POKE #C000 #0000        ; Change contents of location C000 to 0.
! PEEK #C000              ; Display contents of location C000.
  0000                      (Response from Model One.)
```

POLYGN POLYGN

SYNTAX

ASCII	POLYGN npoly, nverts, x, y, ...
FORTRAN Call	CALL POLYGN (NPOLY, NVERT, VERTS)
Binary	[18] [npoly] [highnverts][lownverts] [highx1][lowx1] [highy1][lowy1]...

18 decimal = 22 octal = 12 hex

FUNCTION

The POLYGN command draws polygons in image memory in the current pixel value (VREG 0). The total number of polygons is given by npoly. Each polygon is then specified by giving the number of vertices (nverts), followed by the list of vertices. Each vertex is specified by an x and y coordinate value.

All of the vertices are relative to the current point. The current point is unchanged by the POLYGN command.

Because the POLYGN command allows specification of more than one polygon at a time, the Model One supports arbitrary polygons with interior holes. The polygons may be drawn filled or unfilled, by setting the primitive fill flag with the PRMFIL command. With PRMFIL ON, the nested interior polygons are drawn unfilled; the surrounding polygon is filled.

Note: The number of vertices the POLYGN command accepts depends on how your system's RAM is configured. The default configuration allows up to 251 vertices for unfilled polygons. If all RAM were allocated for polygon fills, you could specify up to 2043 vertices for unfilled polygons. You can reconfigure RAM with the CONFIG command, and display the current configuration with the MAP command.

ASCII PARAMETERS

npoly	8-bit parameter specifying the number of polygons to draw; range is 0 to 255.
nverts	16-bit parameter specifying the number of vertices for each polygon. The range depends on the RAM configuration, as noted above.
x, y	16-bit parameters specifying the x and y coordinates for each vertex; range is -32,768 to 32,767.

POLYGN POLYGN

FORTRAN PARAMETERS

INTEGER*2 NPOLY, NVERT(1), VERTS(2,1)

NPOLY Number of polygons to be drawn.

NVERT One dimensional array containing the number of vertices in each polygon.

VERTS Two dimensional array containing the dx and dy displacements from the current point of all vertices in the defined polygons. VERTS(1,j) contains the jth dx value. VERTS(2,j) contains the jth dy value. If more than one polygon is being defined, the vertices describing the second polygon should be placed in the array immediately after the vertices describing the first polygon.

EXAMPLE

```
! MOVABS 0 0              ; Move current point to 0,0.

! POLYGN  1 3 10,10  10,-10  -10,-10

                          ; Draw a triangle.
! MOVABS 100 100          ; Move current point to 100,100.

! POLYGN 1 3 10,10  10,-10  -10,-10

                          ; Draw same triangle as before, but with vertices
                            relative to 100,100.

! MOVABS -100 100         ; Move current point to -100,100.

! POLYGN 1 4 -30,30  30,30  -30,-30  30,30

                          ; Draw a 4-sided polygon.

! MOVABS -100 100         ; Move current point to -100,100.

! PRMFIL ON               ; Select filled primitives.

! POLYGN 2 3 10,-10  10,-10  -10,-10  4 20,20  20,-20  -20,-20  -20,20

                          ; Draw a triangle nested inside a rectangle.
                            The rectangle is filled and the triangle is not.
```

POP POP

ASCII SYNTAX

```
POP  item, arg      (general form)

POP  CREG, creg          (coordinate register)
POP  VREG, vreg          (value register)
POP  XFORM, xformtype    (transformation matrix)
POP  VAR, vartype        (system variable or flag)
```

FORTRAN Calls

```
CALL  POP (TYPE, CREG, VREG, XFORM, SYSVAR, FPREG)    (general form)

CALL POP0 (CREG)
CALL POP1 (VREG)
CALL POP2 (XFORM)
CALL POP3 (SYSVAR)
```

Binary

```
[118] [item = 0] [creg]          (3 bytes)
[118] [item = 1] [vreg]          (3 bytes)
[118] [item = 2] [xformtype]     (3 bytes)
[118] [item = 3] [vartype]       (3 bytes)

118 decimal = 166 octal = 76 hex
```

POP POP

FUNCTION

The POP command restores the item that you specify by popping it off the stack. The POP command is the inverse of the PUSH command, which pushes items onto the stack.

You must pop stacked items in the exact opposite order as they were pushed.

The PUSH and POP commands perform stack overflow and underflow checking. An error is generated if you exceed the stack bounds.

No type checking is performed by the PUSH and POP commands. Invalid parameter values may produce unpredictable results.

The items that you can push and pop include

- the contents of registers
  - coordinate registers
  - value registers

- transformation matrices
  - current 2-D transformation
  - current 2-D transformation and WCS current point

- system variables or flags
  - primitive fill flag
  - vector pattern
  - pixel function
  - first pixel flag
  - z-buffer function
  - shading mode.

POP POP

ASCII PARAMETERS

item The item to be popped (1 byte)

0 = CREG	Coordinate register.
1 = VREG	Value register.
2 = XFORM	Transformation.
3 = VAR	System variable or flag.

creg Coordinate register number (1 byte). You can use mnemonics for CREGs 0-6, 9, and 10. Refer to the table at the end of this Command Reference.

vreg Value register number (1 byte). You can use mnemonics for VREGs 0 through 3. Refer to the table at the end of this Command Reference.

xformtype Transformation type (1 byte).

0 = XF2D	2-D transformation.
1	Reserved.
2 = XF2DCP	2-D transformation and the WCS current point.

vartype System variable or flag (1 byte).

0 = PRMFIL	Primitive fill flag.
1 = VECPAT	Vector pattern.
2	Reserved
3 = PIXFUN	Pixel function.
4 = FIRSTP	First pixel flag.
5 = ZFUNCT	Depth buffer flag.
6 = SHMODE	Shading mode.

FORTRAN PARAMETERS

INTEGER*2 TYPE, CREG, VREG, XFORM, SYSVAR, FPREG

Note: The FPREG parameter does not apply to the Model One/80.

POP POP

EXAMPLE

```
! XFORM2D ABS 0 .5 0 0 .5 0 0

                          ; Define an absolute 2-D transformation which
                            scales by one half.
! SEGREF 1                ; Draw geometry scaled by one half.
! PUSH XFORM XF2DCP       ; Push the current 2-D transformation and the
                            WCS current point onto the stack.

! XFORM2D REL 0 .707 .707 -.707 .707

                          ; Define a relative 2-D transformation which
                            rotates by 45 degrees.  This transformation
                            is concatenated with the current transformation.
! SEGREF 1                ; Draw geometry scaled by one half and rotated
                            by 45 degrees.
! POP XFORM XF2DCP        ; Pop the half scale transformation which was
                            saved on the stack.
! SEGREF 1                ; Draw geometry scaled by one half and not
                            rotated.
```

PORTO PORTO

SYNTAX

ASCII	PORTO portname, string
FORTRAN Call	CALL PORTO (PORTNAME, STRLEN, STRING)
Binary	[173] [portname] [strlen] ([char1] [char2]...[charn])
	(3 + strlen bytes)
	173 decimal = 255 octal = AD hex

FUNCTION

The PORTO command outputs a text string to the selected port.

If the command is entered in ASCII mode from the local alphanumeric terminal or keyboard, the string to be output is the set of ASCII characters remaining on the command line. The string is terminated by a carriage return (<CR>), but the CR is not sent as part of the string.

If the command is not sent in ASCII mode, then the first byte of the string contains the number of characters in the string (strlen followed by strlen bytes containing the ASCII characters output).

Nonprintable characters can be included in PORTO text strings. Characters with the high bit set or certain control characters (e.g., CTRL-S) can be put into string arguments in ASCII mode by using the backslash (\) to indicate that the immediately following characters are the numeric value of the desired character:

\ddd Indicates 3 decimal digits representing the value of the desired character.

\#hh Indicates the 2 hexadecimal digits representing the desired character.

\\ Indicates the backslash character (\) itself.

PORTO PORTO

ASCII PARAMETERS

portname — Eight-bit parameter which specifies the serial port to receive the text string:

portname = 0 or DMA	(not currently implemented)
portname = 1 or KEYB	(keyboard serial port)
portname = 2 or TABL	(tablet serial port)
portname = 3 or ALPHA	(alpha serial port)
portname = 4 or HOST	(host serial port)

string — The text string to be output.

FORTRAN PARAMETERS

```
CALL PORTO (PORTNAME, STRLEN, STRING)

INTEGER*2     PORTNAME, STRLEN, STRING(1)
```

STRING is an integer array with two characters packed per 16-bit word, as in FORTRAN A2 format.

EXAMPLE

```
! PORTO 1 ABCDEF 1 2 3     ; Output text string "ABCDEF 1 2 3" to the key-
                             board port.
```

PRMFIL PRMFIL

SYNTAX

ASCII	PRMFIL flag
FORTRAN Call	CALL PRMFIL (IFLAG)
Binary	[31] [flag] (2 bytes)
	31 decimal = 037 octal = 1F hex

FUNCTION

The PRMFIL command sets the primitive fill flag to indicate whether 2-D graphics primitives are drawn filled or unfilled. When flag = 1 or ON, any subsequent graphics primitives are drawn filled. The current pixel value is used for all interior pixels, as well as for the perimeter. When flag = 0 or OFF, just the perimeter is drawn in the current pixel value.

The graphics primitives which are affected by the PRMFIL flag are: ARC, CIRCI, CIRCLE, CIRCXY, POLYGN, RECREL, RECTAN, and RECTI.

You can use the VECPAT command to set patterned area and primitive fills.

ASCII PARAMETER

flag	Flag = 1 or ON enables filling; flag = 0 or OFF disables filling.

FORTRAN PARAMETER

INTEGER*2 IFLAG

EXAMPLE

```
! LUT8 1 0,0,255          ; Set the color out for LUT index 1 to blue.
! VAL8 1                  ; Set current pixel value to 1 (blue).
! PRMFIL ON               ; Select filled primitives.
! CIRCLE 50               ; Draw filled blue circle.
! MOVABS 100 100          ; Move current point to 100,100.
! PRMFIL OFF              ; Select unfilled primitives.
! CIRCLE 50               ; Draw unfilled blue circle.
```

PUSH PUSH

ASCII SYNTAX

```
PUSH  item, arg      (general form)

PUSH  CREG, creg          (coordinate register)
PUSH  VREG, vreg          (value register)
PUSH  XFORM, xformtype    (transformation matrix)
PUSH  VAR, vartype        (system variable or flag)
```

FORTRAN Calls

```
CALL  PUSH (TYPE, CREG, VREG, XFORM, SYSVAR, FPREG)    (general form)

CALL PUSH0 (CREG)
CALL PUSH1 (VREG)
CALL PUSH2 (XFORM)
CALL PUSH3 (SYSVAR)
```

Binary

```
[117] [item = 0] [creg]          (3 bytes)
[117] [item = 1] [vreg]          (3 bytes)
[117] [item = 2] [xformtype]     (3 bytes)
[117] [item = 3] [vartype]       (3 bytes)

117 decimal = 165 octal = 75 hex
```

PUSH PUSH

FUNCTION

The PUSH command saves the specified item by pushing it onto the stack. You can then restore the item with the POP command.

You must pop stacked items in the exact opposite order as they were pushed.

The PUSH and POP commands perform stack overflow and underflow checking. An error is generated if you exceed the stack bounds.

No type checking is performed by the PUSH and POP commands. Invalid parameter values may produce unpredictable results.

The items that you can push and pop include

- the contents of registers
  - coordinate registers
  - value registers

- transformation matrices
  - current 2-D transformation
  - current 2-D transformation and WCS current point

- system variables or flags
  - primitive fill flag
  - vector pattern
  - pixel function
  - first pixel flag
  - z-buffer function
  - shading mode.

PUSH PUSH

ASCII PARAMETERS

item The item to be pushed.

0 = CREG	Coordinate register.
1 = VREG	Value register.
2 = XFORM	Transformation.
3 = VAR	System variable or flag.

creg Coordinate register number (1 byte). You can use mnemonics for CREGs 0-6, 9, and 10. Refer to the table at the end of this Command Reference.

vreg Value register number (1 byte). You can use mnemonics for VREGs 0 through 3. Refer to the table at the end of this Command Reference.

xformtype Transformation type (1 byte).

0 = XF2D	2-D transformation.
2 = XF2DCP	2-D transformation and the WCS current point.

vartype System variable or flag (1 byte).

0 = PRMFIL	Primitive fill flag.
1 = VECPAT	Vector pattern.
2	Reserved.
3 = PIXFUN	Pixel function.
4 = FIRSTP	First pixel flag.
5 = ZFUNCT	Depth buffer flag.
6 = SHMODE	Shading mode.

FORTRAN PARAMETERS

INTEGER*2 TYPE, CREG, VREG, XFORM, SYSVAR, FPREG

Note: The FPREG parameter does not apply to the Model One/80.

EXAMPLE

```
! XFORM2D ABS 0 .5 0 0 .5 0 0

                          ; Define an absolute 2-D transformation which
                            scales by one half.
! SEGREF 1                ; Draw geometry scaled by one half.
! PUSH XFORM XF2DCP       ; Push the current 2-D transformation and the
                            WCS current point onto the stack.

! XFORM2D REL 0 .707 .707 -.707 .707

                          ; Define a relative 2-D transformation which
                            rotates by 45 degrees.  This transformation
                            is concatenated with the current transformation.
! SEGREF 1                ; Draw geometry scaled by one half and rotated
                            by 45 degrees.
! POP XFORM XF2DCP        ; Pop the half scale transformation which was
                            saved on the stack.
! SEGREF 1                ; Draw geometry scaled by one half and not
                            rotated.
```

QUIT QUIT

SYNTAX

ASCII	QUIT
FORTRAN Call	CALL QUIT
Binary	[255] (1 byte)
	255 decimal = 377 octal = FF hex

FUNCTION

The QUIT command exits Graphics mode and returns to Alpha mode. This command should be used to return to Alpha mode when the host or local alphanumeric terminal is finished issuing graphics commands.

If the Model One has been configured for pure ASCII format (with the ASCII command), the system is restored to binary or ASCII hex format, as set with the SYSCFG command.

If the DMA port was in Graphics mode, and characters had been sent from the host over a serial line, these characters would now be displayed on the terminal as a result of the QUIT command.

RDCNFG RDCNFG

SYNTAX

ASCII RDCNFG

FORTRAN Call CALL RDCNFG (IARRAY)

Binary [208] (1 byte)

208 decimal = 320 octal = D0 hex

FUNCTION

The RDCNFG command returns the hardware and firmware configuration of the Model One/80 to the port which is in Graphics mode. The following information is returned. If the alphanumeric terminal is the current port, the data is formatted as hexadecimal words separated by two spaces.

Type of Information	Number of Words	Values
Type of Model One	1	0000: Model One/80 with 4K WCS 0001: Model One/80 with 8K WCS 0002: Model One/80 with 16K WCS
Firmware options	1	0000: No firmware options 0001: Shading 0002: Display List 0004: Tektronix 4014 emulator 0008: Interactive device support 00FF: All firmware options Note: Firmware options are additive. 00FF is equivalent to 000F.
Hardware options	1	0000: Not currently implemented for Model One/80.
RAM blocks available for macros, downloaded text, display lists, areafill/ polyfill/zpatch scratch, alpha windows	1	0010: 16 Kbytes
Reserved	1	0000

RDCNFG RDCNFG

Type of Information	Number of Words	Values
Firmware revision and version. Format is RRVV where RR is revision, and VV is version.	1	Example: 0100 is version 1.0
Image memory size (x,y)	2	04FF: 1279 decimal 03FF: 1023 decimal
Visible screen size (x,y)	2	04FF: 1279 decimal 03FF: 1023 decimal
Memory unit configuration	16	See explanation below

The following word displays the currently selected emulation (high byte) and graphics input device (low byte).

Currently selected emulation	(high byte)	0 (decimal): no emulator in use (alpha terminal) 1 (decimal): alpha emulator 2 (decimal): VT100K emulator 3 (decimal): VT100C emulator 4 to 255 (decimal): reserved
Graphics input device	(low byte)	0 (decimal): GTCO (old) 1 (decimal): GTCO 2 (decimal): SUMMA 3 (decimal): RMOUSE 4 (decimal): UGRID 5 to 255 (decimal): reserved

The following 6 words display the current special characters:

Word 1	1	high byte: ENTRA low byte: BREAK
Word 2	1	high byte: WARM low byte: KILL
Word 3	1	high byte: BS (backspace) low byte: ACK

RDCNFG RDCNFG

Type of Information	Number of Words	Values
Word 4	1	high byte: ABORT low byte: DEBUG
Word 5	1	high byte: XON low byte: XOFF
Word 6	1	high byte: NEW low byte: reserved
Reserved	17	0000

Memory Unit Configuration

The 16 words used for displaying the memory unit configuration are shown below, for 8-bit systems, 24-bit systems, and 24-bit systems with a depth buffer.

8-Bit System

FF00 0000 0000 0000 0000 ... 0000

24-Bit System

FF00 FF00 FF00 0000 0000 ... 0000

24-Bit System with a Depth Buffer

FF00 FF00 FF00 FFFF 0000 ... 0000

RDCNFG RDCNFG

FORTRAN PARAMETER

```
INTEGER*2     IARRAY(1)
```

IARRAY is a word (16-bit) array.

IARRAY(1)	Type of Model One.
IARRAY(2)	Firmware options.
IARRAY(3)	Hardware options (not implemented).
IARRAY(4)	RAM blocks available.
IARRAY(5)	Reserved.
IARRAY(6)	Firmware revision.
IARRAY(7)	Image memory size (x).
IARRAY(8)	Image memory size (y).
IARRAY(9)	Screen size (x).
IARRAY(10)	Screen size (y).
IARRAY(11) to IARRAY(26)	Memory unit configuration.
IARRAY(27)	Currently selected emulation (high byte) and graphics input device (low byte)
IARRAY(28)	ENTRA (high byte) and BREAK (low byte) special characters
IARRAY(29)	WARM (high byte) and KILL (low byte) special characters
IARRAY(30)	BS (backspace) (high byte) and ACK (low byte) special characters
IARRAY(31)	ABORT (high byte) and DEBUG (low byte) special characters
IARRAY(32)	XON (high byte) and XOFF (low byte) special characters
IARRAY(33)	NEW special character (high byte) and reserved (low byte)
IARRAY(34) to IARRAY(50)	Reserved

RDKEYB RDKEYB

SYNTAX

ASCII	RDKEYB nchars, term, flags
FORTRAN Call	CALL RDKEYB (NCHARS, TERM, FLAGS, IARRAY)
Binary	[241] [nchars] [term] [flags] (4 bytes)
	241 decimal = 361 octal = F1 hex

FUNCTION

The RDKEYB command reads back a line of characters from the alphanumeric terminal keyboard and sends it to the port in Graphics mode. This command is primarily used for DMA systems to simulate a serial terminal environment.

The nchars parameter sets the number of characters to be read before sending the buffer of characters to the port in Graphics mode. The term parameter sets the terminator character. The flags parameter specifies whether the characters are echoed and whether a terminator is expected.

Note: There is no default terminator character.

If no terminator is expected, the number of characters specified by nchars must be typed before the buffer will be sent to the port in Graphics mode.

How the Model One Returns the Keyboard Buffer: For DMA

In binary mode (RDMODE = 1), the number of bytes returned will be the number specified by the nchars parameter plus two. The first byte in the buffer will indicate the number of characters actually read, not including the terminator. The second byte gives the terminator character (0 if no terminator was used). The rest of the buffer includes the keyboard characters, without the terminator, and any nulls used to pad the buffer. An even number of bytes is always returned, regardless of the nchars value. The bytes are always returned in binary.

If nchars is set to 0, only the first and second byte are sent. Setting nchars to zero can, therefore, be used to determine if anything has been typed, without actually reading back the buffer until it is needed.

FUNCTION, continued

When the terminator is returned (the second byte sent back), the high bit is set to indicate the presence of the terminator. If no terminator character is present, the high bit of the second byte will be zero (the high bit will not be set). Thus, even a null can be used as a terminator. See examples below.

How the Model One Returns the Keyboard Buffer: For Serial

In ASCII mode (RDMODE = 0), the number of bytes returned will be the number specified by the nchars parameter plus 6 bytes. The first three bytes in the buffer represent the ASCII decimal representation of the number of characters actually read (similar to the FORTRAN I3 format). The next 3 bytes represent the ASCII decimal representation of the terminator character. For example, if the terminator character was 108, the first three bytes would be 49 48 56. The rest of the buffer includes the keyboard characters, without the terminator, and any nulls used to pad the buffer.

If nchars is set to 0, only the first six bytes are sent. Setting nchars to zero can, therefore, be used to determine if anything has been typed, without actually reading back the buffer until it is needed.

When the terminator is returned (the second set of 3 bytes sent back), the high bit of the value of the terminator is set, prior to encoding in ASCII decimal, to indicate the presence of the terminator. In the example above, the value returned is 235, which would be transmitted as 50 51 54. If no terminator character is present, the set of 3 bytes will be zero (the high bit will not be set). Thus, even a null can be used as a terminator. See examples below.

ASCII PARAMETERS

nchars — Eight-bit parameter which specifies the number of characters to be read before sending the buffer to the port in Graphics mode. The buffer will be padded with nulls and sent before the number of characters specified with nchars have been typed if the flags parameter specifies that a terminator is expected and the terminator is typed.

term — The terminator character; valid character numbers are 0 to 127. The character list is given with the TEXTDN command. There is no default value.

ASCII PARAMETERS, continued

flags — Eight-bit parameter which specifies whether echoing should occur and whether a terminator is expected.

	Echo Typed Characters?	Expect Terminator?
flags = 0	Yes	Yes
flags = 1	No	Yes
flags = 2	Yes	No
flags = 3	No	No

FORTRAN PARAMETERS

```
CALL RDKEYB (NCHARS, TERM, FLAGS, IARRAY)

INTEGER*2     NCHARS, FLAGS
LOGICAL*1     TERM, IARRAY(1)
```

IARRAY contains the number of characters transmitted (IARRAY(1)), the terminator character (IARRAY(2)), and the data read from the keyboard upon return (IARRAY(3): IARRAY(3 + NCHARS))

EXAMPLE

```
! RDKEYB 10 13 0        ; Start RDKEYB command (13 = 0D hex, specifying a
                          carriage return.
1 2 3 4 5 6 7 8 9 0     ; User types this sequence at the keyboard.

0A 00 31 32 33 34 35 36 37 38 39 30

                        ; Model One returns 12 hex bytes; byte 1 indi-
                          cates 10 bytes in keyboard buffer, byte 2 indicates
                          no terminator encountered, and subsequent bytes
                          hold the keyboard buffer (in hex).

1 2 3 4 <CR>            ; User types this sequence (including a carriage re-
                          turn (<CR>).

04 8D 31 32 33 34 00 00 00 00 00 00

                        ; Model One returns 4 characters in the buffer;
                          byte 2 indicates <CR> encountered, and subsequent
                          bytes are keyboard buffer padded with nulls.
```

EXAMPLE, continued

```
! RDKEYB 0 13 0          ; Start RDKEYB command.
1 2 3 4 5 6 7 8 <CR>     ; Host issues RDKEYB only when characters are typed.

08 8D                    ; Model One returns 2 bytes; byte 1 indicates
                           there are 8 bytes in the buffer, and byte 2 indi-
                           cates that terminator <CR> was found.
```

RDMASK RDMASK

SYNTAX

ASCII	RDMASK mask
FORTRAN Call	CALL RDMASK (MASK)
Binary	[158] [mask] (2 bytes)
	158 decimal = 236 octal = 9E hex

FUNCTION

The RDMASK command sets the read mask for the Model One's image memory planes. The 8-bit read mask is ANDed with the data from image memory immediately before the data enters the look-up table. If a bit is set (1), the corresponding bit plane is read-enabled; if the bit is cleared (0), the corresponding bit plane is not displayed.

On an 8-bit system, the read mask corresponds directly to the eight bit planes of image memory. On an 8-bit system, you can also set the read mask with the RMSK16 commands.

On a 24-bit system with RGBTRU ON, the 8-bit read mask applies in the same way to the red, green, and blue image memory banks.

The read mask may be used in conjunction with the write mask to store multiple images in image memory and select them for display. The WRMASK command sets the write mask.

The RDMASK and WRMASK commands can also be used for double buffering. The image memory on each memory board is divided into two 4-bit banks. If you use the WRMASK command to write enable bit planes in only one of these 4-bit banks, and the RDMASK command to read enable bit planes in only the other 4-bit bank, then the graphics processor will have full memory bandwidth into the selected bit planes.

ASCII PARAMETER

mask The 8-bit read mask; range is 0 to 255.

FORTRAN PARAMETER

INTEGER*2 MASK

RDMODE RDMODE

SYNTAX

ASCII	RDMODE flag
FORTRAN Call	CALL RDMODE (FLAG)
Binary	[211] [flag] (2 bytes)
	211 decimal = 328 octal = D3 hex

FUNCTION

The RDMODE command sets the format for reading back data to the host computer in response to a readback command. The two modes are ASCII decimal (flag = 0 or OFF) and binary (flag = 1 or ON). ASCII decimal format should be used for serial communications. Binary format should be used for DMA communication.

When ASCII decimal data is sent, it is followed by a carriage return. When binary data is sent, no carriage return is included.

The commands which read back information are: READBU, READCR, READER, READP, READVR, READW, and READWE. The display list readback commands are: RDPICK, RDREG, RDXFORM, SEGINQ, and SYSTAT. RDKEYB reads back the keyboard, but is unaffected by the RDMODE setting.

ASCII PARAMETER

flag	Flag = 0 or OFF, readback in ASCII decimal (default); flag = 1 or ON, readback in binary.

FORTRAN PARAMETER

INTEGER*2	FLAG

RDPICK RDPICK

SYNTAX

ASCII	RDPICK type, startent, totalent
FORTRAN Call	CALL RDPICK (TYPE, START, NUM, PICENC, PICREL, PICIDS, SEGIDS, DEPTH, TREES)
Binary	[239] [type] [startent] [totalent] (6 bytes)
	239 decimal = 357 octal = EF hex

FUNCTION

The RDPICK command reads back information which is stored in the pick buffer. The command also updates the segment ID register (SEGREG) and the pick ID register (PIDREG) with the last entry in the pick buffer that is read back.

The type parameter specifies the type of information to read back. You can read back the following six types of information.

- The number of pick hits encountered and the number of pick hits recorded (this information is taken from counters; SEGREF and PIDREG are not updated).
- Pick ID numbers of specified pick hits (PIDREG is updated with last entry read back).
- Segment ID numbers (SEGREG is updated).
- Segment ID/pick ID pairs (SEGREG and PIDREG are updated).
- Unpacked trees of nested segment IDs and pick IDs (SEGREG and PIDREG are updated).
- Packed trees of nested segment IDs and pick IDs (SEGREF and PIDREG are updated).

If RDPICK fails to identify any geometry, it returns -1 as the segment ID and the pick ID.

The startent parameter specifies the pick buffer entry to start reading from. If you specify 0, the Model One starts reading back from the next unread location.

FUNCTION, continued

The totalent parameter specifies the total number of entries to read back. If you specify 0, the Model One updates the registers with the specified entry (startent), but does not return any data.

The following are some examples of updating the registers without returning data.

- RDPICK PICKID 0 0 — Update the pick ID register with the next unread entry in the pick buffer.
- RDPICK SEGID 1 0 — Update the segment ID register with the first entry in the pick buffer.
- RDPICK SEGPID 2 0 — Update both registers with the second entry in the pick buffer.

Note: The parameter you select with this command (except for RDPICK PICKS) must correspond to a parameter you previously specified with SETGL PICKBUF. The specific type of information you ask for with RDPICK is stored in the pick buffer and therefore available for readback only after an appropriate choice with SETGL PICKBUF. For example, SETGL PICKBUF PICKID does not save pick information on segment IDs or tree hierarchies, whereas SETGL PICKBUF TREE saves pick information on segment IDs, pick IDs, and tree hierarchies. Without your selecting a correct correspondence, the pick buffer will not contain the data you request and it therefore will return minus ones. This return could be confused with "no pick hits" or "no pickable entities" when, in fact, there was a hit.

ASCII PARAMETERS

type — Type of information to be returned (8 bits).

0 = PICKS	Number of pick hits encountered (16 bits) and number of pick hits recorded (16 bits).
1 = PICKID	An array of pick ID numbers (32 bits).
2 = SEGID	An array of segment ID numbers (32 bits).
3 = SEGPID	An array of segment ID/pick ID pairs (32 bits).
4 = TREEU	Unpacked trees of nested segment ID/pick ID pairs (32 bits).
5 = TREEP	Packed trees of nested segment ID/pick ID pairs (32 bits).

ASCII PARAMETERS, continued

startent Integer which specifies the pick buffer entry to start reading from (16 bits).

With type 0 (PICKS), specify 0 (parameter is not relevant).

With types 1-5, specifying 0 instructs the Model One to start reading back from the next unread location.

totalent Integer which specifies the total number of entries to read back (16 bits).

With type 0 (PICKS), specify 0 (parameter is not relevant).

With types 1-5, specifying 0 updates the registers without returning data.

FORTRAN PARAMETERS AND SUBROUTINE CALLS

```
CALL RDPICK (TYPE,START,NUM,PICENC,PICREL,PICIDS,SEGIDS,DEPTH,TREES)

Input Parameters:   INTEGER*2     TYPE, START, NUM

Output Parameters:  INTEGER*2     PICENC, PICREC
                    INTEGER       PICIDS(1), SEGIDS(1), DEPTH(1)
                    INTEGER*4     TREES(1,1)
```

The following subroutines are provided for convenience. These subroutines correspond to the ASCII types 0 through 5.

```
RDPCK0 (PICENC, PICREC)
RDPCK1 (START, NUM, PICIDS)
RDPCK2 (START, NUM, SEGIDS)
RDPCK3 (START, NUM, SEGIDS, PICIDS)
RDPCK4 (START, NUM, DEPTH, UTREES)
RDPCK5 (START, NUM, DEPTH, PTREES)
```

EXAMPLE

The following three examples illustrate many of the different ways that you can use the RDPICK command.

First we define Segment 1, which is a truck including a body, chassis, and two wheels. (This is the same segment that is defined in the SEGDEF command example, and is repeated here for easy reference.)

Following the segment definition, we set the center of the pick aperture, execute the segment in pick mode, and then execute the RDPICK command to read back information. We repeat this procedure three times with the pick aperture set in different locations.

Segment Definition

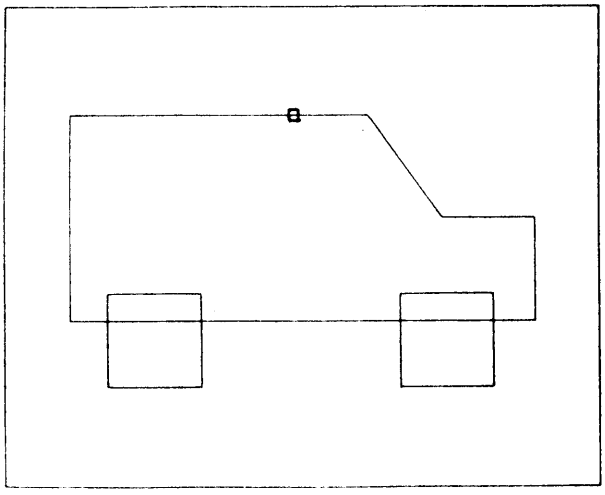
```
! SEGINI 0 256          ; Initialize display list structures with a
                          block size of 256 bytes.
! SEGDEF 1              ; Begin definition of Segment 1, which defines
                          the truck body, chassis, and the position of
                          the 2 wheels.
$ PICKID 10             ; Assign pick ID 10 label to the truck body.
$ PUSH XFORM XF2DCP     ; Push current 2-D transformation and WCS current
                          point onto the stack.
$ MOVABS -500 -160      ; Move current point to lower left corner of truck.
$ DRWREL 0 440          ; Draw 5 vectors to define truck body.
$ DRWREL 640 0
$ DRWREL 160 -220
$ DRWREL 200 0
$ DRWREL 0 -220
$ PICKID 20             ; End pick ID 10 (body); assign pick ID 20 label to
                          the chassis (bottom line) of truck.
$ DRWREL -1000 0        ; Draw chassis of truck.
$ MOVABS -320 -200      ; Move current point to position rear wheel.
$ SEGREF 2              ; Nest Segment 2, the wheel.
$ MOVABS 310 -200       ; Move current point to position front wheel.
$ SEGREF 2              ; Nest Segment 2, the wheel.
$ POP XFORM XF2DCP      ; Pop current 2-D transformation and WCS current
                          point from the stack.
$ SEGEND                ; End pick ID 20 (chassis); end definition of
                          Segment 1.

! SEGDEF 2              ; Begin definition of Segment 2, which defines the
                          shape of the wheel.
$ PICKID 30             ; Assign pick ID 30 label to the wheel.
$ PUSH CREG CURPNT      ; Save current point (center of wheel).
$ MOVREL -100 -100      ; Move current point to left corner of wheel.
$ RECREL 200 200        ; Draw wheel.
$ POP CREG CURPNT       ; Restore current point (center of wheel).
$ SEGEND                ; End pick ID 30 (wheel); end definition of
                          Segment 2, the wheel.
```

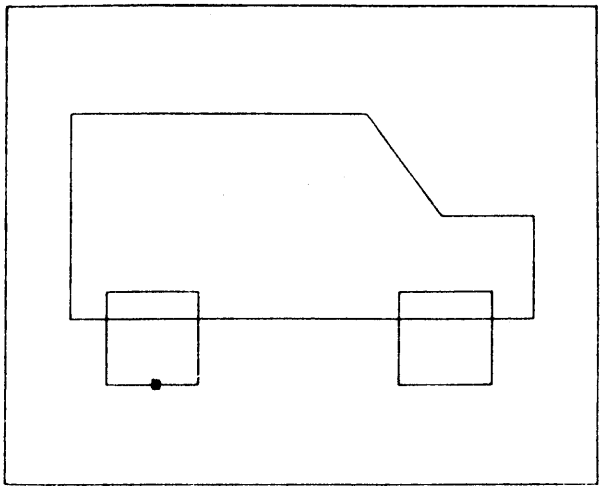
Drawing of Segment 1 with Pick Aperture Locations

The figures below show Segment 1, and the three different locations where we will set the pick aperture in the three examples which follow.

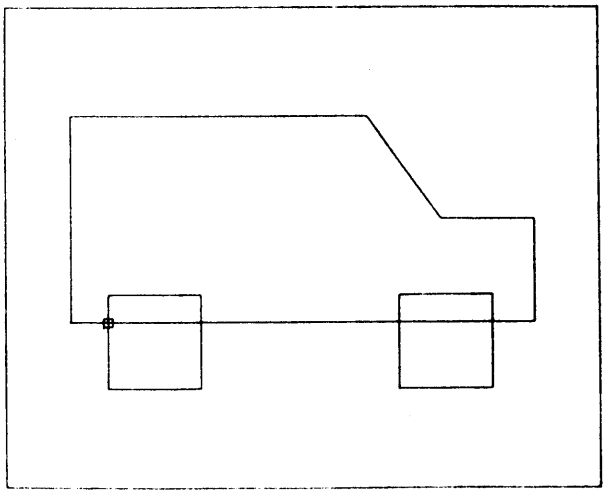
EXAMPLE 1: (0,280)



EXAMPLE 2: (-300,-300)



EXAMPLE 3: (-420,-160)



EXAMPLE 1: Reading Back One Pick Hit in Top Level Segment

```
! SETGL PICKAP 10 10      ; Set the pick aperture size to 20 x 20.
! SETGL PICKBUF TREE      ; Specify to save entire trees in pick buffer.
! CLOAD 19 0,280          ; Specify the center of pick aperture. (See the
                           preceding figure labelled EXAMPLE 1.)
! EXMODE PICK             ; Set execution mode to pick.
! SEGREF 1                ; Execute (pick) Segment 1.
! RDPICK PICKS 0 0        ; Read back number of pick hits encountered and
                           number of pick hits recorded; registers are not
                           updated.
  00001 00001             ; The Model One returns:  1 pick hit encountered
                           and recorded.
! RDPICK PICKID 1 1       ; Read back pick ID; update PIDREG with pick ID.
 0000000010                ; The Model One returns: pick ID is 10.
! RDPICK SEGID 1 1        ; Read back segment ID; update SEGREG.
 0000000001                ; The Model One returns: segment ID is 1.
! RDPICK SEGPID 1 1       ; Read back segment ID/pick ID pairs; update
                           PIDREG and SEGREG.
 0000000001 0000000010     ; The Model One returns: segment ID is 1, pick ID
                           is 10.
! RDPICK TREEU 1 1        ; Read back the tree hierarchy in unpacked format,
                           update PIDREG and SEGREG.
  00001
 0000000001 0000000010
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001
 -000000001 -000000001    ; The Model One returns: one tree hit, nesting
                           level is one deep, the top level is Segment 1,
                           the hit was on graphics labelled pick ID 10.

! RDPICK TREEP 1 1        ; Read back the tree hierarchy in packed format;
                           update SEGREG and PIDREG.
  00001                   ; The Model One returns the same information as
 0000000001 0000000010     for the packed format, but does not read back
                           the minus ones.
! EXMODE NORMAL           ; Restore execution mode to normal draw.
```

EXAMPLE 2: Reading Back One Pick Hit in Nested Segment

```
! SETGL PICKBUF TREE      ; Specify to save entire trees in pick buffer.
! CLOAD 19 -300 -300      ; Specify the center of the pick aperture.  (See
                           the preceding figure labelled EXAMPLE 2.)
! EXMODE PICK             ; Set execution mode to pick.
! SEGREF 1                ; Execute (pick) Segment 1.
! RDPICK PICKS 0 0        ; Return number of pick hits encountered and
                           recorded.
  00001 00001             ; The Model One returns:  1 pick hit encountered
                           and recorded.
! RDPICK SEGPID 1 1       ; Read back segment ID/pick ID pairs; update
                           SEGREG and PIDREG.
 0000000002 0000000030     ; The Model One returns: segment ID is 2, pick ID
                           is 30.
! RDPICK TREEP 1 1        ; Read back tree hierarchy in packed format.
  00002                    ; The Model One returns:  one tree hit, nesting
 0000000001 0000000020     level is two deep.  The top level is Segment 1,
 0000000002 0000000030     pick ID 20.  The second level is Segment 2,
                           pick ID 30.
! EXMODE NORMAL           ; Restore execution mode to normal draw.
```

EXAMPLE 3: Reading Back Two Pick Hits

```
! SETGL PICKBUF TREE     ; Specify to save entire trees in pick buffer.
! CLOAD 19 -420 -160     ; Set center of pick aperture.  (See preceding
                          figure labelled EXAMPLE 3.)
! EXMODE PICK            ; Set execution mode to pick.
! SEGREF 1               ; Execute (pick) Segment 1.
! RDPICK PICKS 0 0       ; Read back number of pick hits encountered and
                          recorded; registers are not updated.
  00002 00002            ; Model One returns:  2 pick hits encountered and
                          recorded.
! RDPICK SEGPID 1 2      ; Read back Segment ID/pick ID pairs, starting with
                          first entry and reading two entries; update
                          SEGREG and PIDREG with last entries read back.
 0000000001 0000000020
 0000000002 0000000030
                          ; The Model One returns: first hit is Segment 1,
                          pick ID 20; second hit is Segment 2, pick ID 30.
! RDREG 0                ; Read back contents of SEGREG and PIDREG.
 0000000002 0000000030   ; Registers contain last entry read back.
! RDPICK SEGPID 1 1      ; Read back Segment ID/pick ID pairs, starting with
                          first entry and reading back only one entry.
 0000000001 0000000020
! RDREG 0
 0000000001 0000000020   ; The registers now contain the last entry read
                          back, which is now the first hit.
! RDPICK SEGPID 0 0      ; Update registers with next unread entry in pick
                          buffer.
! RDREG 0
 0000000002 0000000030   ; The registers have been updated with next unread
                          entry.
! RDPICK TREEP 1 2       ; Read back tree hierarchy in packed format;
                          update registers.
  00001                  ; The Model One returns:  two trees hit.  For first
 0000000001 0000000020    tree, nesting level is one deep, the top level is
  00002                   Segment 1, the hit was on graphics labelled
 0000000001 0000000020    pick ID 20.  For second tree, nesting level is
 0000000002 0000000030    two deep, the top level is Segment 1, pick ID 20,
                          and the second level is Segment 2, pick ID 30.
! EXMODE NORMAL          ; Restore execution mode to normal draw.
```

RDPID RDPID

SYNTAX

ASCII	RDPID bf
FORTRAN Call	CALL RDPD80 (MAXSIZ, NVALS, BYTRAY)
Binary	[237] [0] (2 bytes)
	237 decimal = 355 octal = ED hex

FUNCTION

The RDPID command reads back all the commands which have a pick ID equal to PIDREG and a segment ID equal to SEGREG. The RDPID command returns a list of commands which contains opcodes and parameters. The information is in the format (n, nray), where n gives the number of bytes returned and nray is a list of primitives n bytes long. If PIDREG is equal to -1, then all commands in all the pick IDs in the current segment are returned.

For serial communications, the bf parameter (blocking factor) tells the Model One how many elements to send before inserting a carriage return into the output stream. If the end of the window is reached before the block is filled, the block is padded with zeros and sent. After sending each block, the Model One then waits for an ACK (06 hex or 86 hex) character from the host before sending out another block of data. The acknowledge character must be sent from the host as a single 7-bit control character, regardless of whether the host normally sends data to the Model One in 8-bit binary or ASCII hex.

ASCII PARAMETER

bf Blocking factor (8 bits). Range is 1 to 255.

FORTRAN PARAMETERS

Input Parameter:	INTEGER*2	MAXSIZ
Output Parameters:	INTEGER*2	NVALS
	LOGICAL*1	BYTRAY(1)

MAXSIZE The maximum size of the array in bytes.

NVALS The number of elements returned.

BYTRAY Array containing elements returned.