PS 390 RELEASE NOTES

EVANS & SUTHERLAND

April 1987
E&S #904015-605
Rev B

CONTENTS

FIGURES AND TABLES

PS 390 GRAPHICS FIRMWARE RELEASE NOTES

Version A2.V02

1. GENERAL INTRODUCTION

These release notes document functionality of the PS 390 and are intended as a supplement to the *PS 300 Document Set* which describes the operation and programming of the PS 300 line of computer graphics systems. These notes can be placed in the *Document Set* behind the Release Notes tab in Volume 3A.

The PS 390 has a new hardware configuration with a reduced-size cabinet and a Joint Control Processor (JCP) card, which are explained in Section 2.1 of these Notes. The hardware configuration and calligraphic display documented in the *Document Set* are not applicable to the PS 390. New users should note that a different set of peripherals is available with the PS 390 although the peripherals documented in the *Document Set* are also supported under PS 390. Procedures for using the peripherals with the multiplexing box are explained in Section 2.3.

You should assume that all programming information in the *PS 300 Document Set* and accompanying installation manuals (dependent on your particular configuration) is applicable to the PS 390 unless specifically noted in these Release Notes.

Changes and additions to the PS 300 Graphics runtime firmware and host software released since the publication of the *Document Set* are contained in Part II of these Notes. This section consolidates the information from previous release notes that applies to the PS 390, and includes formal change pages for the Command and Function Summaries and for the Graphics Support Routines (GSRs) in the *PS 300 Document Set.* Please discard the old pages in your set and replace with these new pages. Some new pages documenting specific PS 390 functionality and other pages with new information are included and should be inserted in the appropriate place in your *Document Set.*

1.1 Notes to New Users

New users should familiarize themselves with the information in these notes and in the *PS 300 Document Set* and note where PS 390 information contained in this package differs from information in the *PS 300 Document Set.*

1.2 Notes to Current Users

One of the primary concerns in developing the PS 390 runtime firmware was maintaining compatibility with previous systems so that existing PS 300 programs would run on the PS 390 without modification. This was almost completely achieved. However, some incompatibilities exist because Port 2 is no longer used and there is no support for DMR-11 interface or multi-user systems. Also, scope 0 is the only scope enabled; therefore, Set Scope commands (both ASCII and GSR) should not be used.

The PS 390 new hardware configuration with a reduced-size cabinet and a Joint Control Processor (JCP) card is explained in Section 2.1 of these Notes. If you have already upgraded to a reduced-size cabinet, you should have already received this information.

You should also have Release Notes for Version A1.V02 and Version A2.V01 of the graphics runtime firmware. Note that the information contained in those release notes which reflects current functionality has been consolidated and included in Part II of these notes.

Current PS 340 users should note that rendering capability on the PS 390 is available only by ordering the rendering option. Without this option, you can display objects defined as polygons on the PS 390, but you cannot perform any rendering operations. For users with the rendering option, the PS 390 Rendering Option Release Notes are included with the PS 390 Release Notes package.

Current PS 350 users should note that the PS 390 is plug compatible with PS 350 applications with the exceptions given in Section 4 of these Notes. Please note that the light pen is not supported with this release. The nodes created by applications using the Lightpen command (both ASCII and GSR) will be treated as no-operation nodes.

Some PS 390 information contained in this release package is identical to information and change pages contained in the *PS 350 User's Manual.* Please note that information in these release notes supersedes any other documentation explaining similar or identical capabilities of the PS 390.

1.3 Notes to All Users

As previously mentioned, a different set of peripherals is offered with the PS 390 although existing peripherals are still supported. Users of existing peripherals and users with new peripherals must both use a multiplexing box as there is no data concentrator on the display. Procedures for using the new mux boxes are given in Section 2.3 of these Notes. Documentation for the new set of peripherals is supplied as a separate document included with this release package.

Data formats provided for those users with the Parallel Interface or for those who access internal data are provided in the *PS 300 Advanced Programming* guide included with this release package. (PS 350 users please note that PS 390 data formats and PS 350 data formats are identical). No new GSRs routines are provided with this release. GSR routines supporting new PS 390 capability are planned for future releases.

Please take special note of Section 4 of these Release Notes, which documents PS 390 exceptions to existing PS 300 documentation.

Direct your questions and comments to the Evans & Sutherland Customer Engineering Hotline 1-800-582-4375 (except Utah). Within Utah, customers should call 582-9412.

1.4 Release Package Contents

This PS 390 Release Package contains the following items.

- One copy of the Graphics runtime firmware Version A2.V02

  For users with the rendering option, this is on the Visualization diskette. Instructions for loading the firmware are contained in Volume 5 of the *Document Set*. Instructions for configuring your firmware diskette according to which options you have at your installation are contained in section 3 of these Notes.

- PS 390 host software distributed on magnetic tape including (but not limited to) the following:
  - PS 300 Graphics Support Routines (GSRs). The files READFOR.GSR and READPAS.GSR contain descriptions of the FORTRAN and Pascal GSR software.
  - The PS 300 Host-Resident I/O Subroutines
  - Three programming utilities: NETEDIT, NETPROBE, and MAKEFONT (For VAX/VMS users only).
  - Writeback Feature

  Documentation for the Writeback feature is included in this release package. More detail on the GSRs, I/O Subroutines, and programming utilities can be found in Volumes 3 and 4 of the *Document Set*.

- One copy of the Diagnostic Utility Diskette

  This diskette provides all the utility programs described in Volume 5, Section 10 of the *Document Set*. Please refer to that section for instructions on using the utility programs for backup and file management and make note that the new Diagnostic Utility Diskette is the only diskette that should be used to load these programs.

- *PS 390 Raster Programming* guide

  This manual documents how to send run-length encoded pixel data to the PS 390.

- *PS 300 Advanced Programming* guide

  This manual is intended for use by experienced programmers as a guide to writing functions and as a reference for doing direct Physical I/O with the Parallel Interface.

- *PS 390 Peripherals Reference Manual*

1.5 Distribution Tape Format and Installation Procedure

All PS 390 VAX/VMS sites will receive the distribution tape (PS 390 host software) in VMS Backup format. To install the VAX PS 390 host software, first create a subdirectory for the PS 390 software and set your default to that directory. Using the VMS Backup Utility, enter the following commands:

```
$  Allocate MTNN:
$  Mount/Foreign MTNN:
$  Backup MTNN:PSDIST.BCK [...]*.*
$  Dismount MTNN:
$  Deallocate MTNN:
```

where MTNN: is the physical device name of the tape drive being used.

This will create the subdirectory A2V01.DIR which is the parent directory of the PS 390 host software.

UNIX sites will receive a 1600-bpi distribution tape in tar format. IBM sites will receive a 1600-bpi distribution tape with a block size of 6400 and a logical record length of 80.

All PS 390 sites that are not DEC VAX/VMS, UNIX, or IBM, will receive a variable length ANSI format distribution tape containing the PS 390 host software. Consult your system operation manual for instructions on reading ANSI-formatted tapes.

2. INTRODUCTION TO PS 390

The PS 390 provides the real–time interaction capability and line quality of a calligraphic system with the flicker–free images of a raster system, combining the desirable features of both technologies while eliminating the disadvantages of each.

These capabilities were accomplished by the development of several VLSI chips and one custom gate array designed for the real–time manipulation of anti–aliased raster lines matching or exceeding the quality of calligraphic lines.

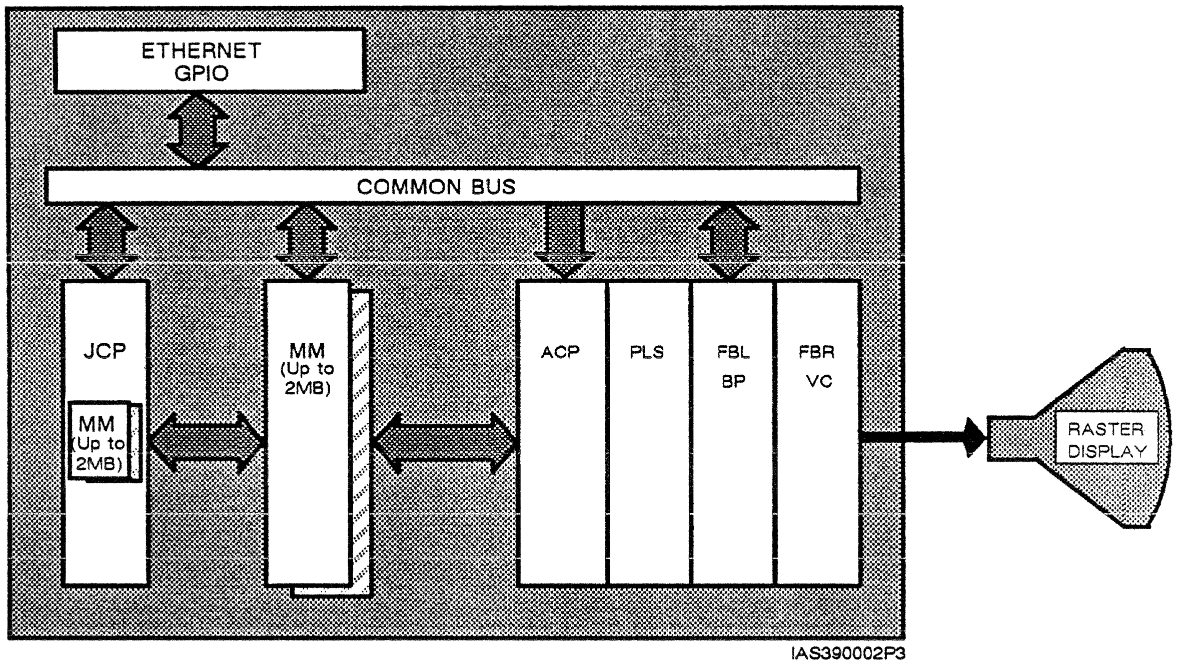
The graphics pipeline of the PS 390 is 32 bits which provides high–precision processing required for large and complex models. The frame buffer is a 48–bit frame buffer, double buffered.

With this version of the firmware, you can use the PS 390 monitor to display host–generated pixel images. The PS 390 accepts raster data in run–length encoded format. A discussion of how to accomplish this is contained in the *PS 390 Raster Programming* manual included with this release package. Existing PS 340 applications using run–length encoding to display host–generated images will run unchanged on the PS 390.

The local capability to create, render, and shade polygonal models on the PS 390 is available with the purchase of the rendering option. With this option, you can display and manipulate a wireframe model in one viewport of the screen and display the same model as a shaded image in another viewport on the screen. Capability now supported on the PS 340 graphics system is supported on the PS 390 with the rendering option. This includes the ability to apply sectioning, back–face removal, and hidden–line rendering operations to wireframe models and to display static images with a wash, flat, Phong, or Gouraud shading style.

2.1 System Hardware Overview

The PS 390 is housed in a new reduced size cabinet and contains a new Joint Control Processor (JCP) card. The JCP replaces the Graphics Control Processor, up to two mass memory cards, and (optionally) the PS 300 IBM 3278 GPIO card. The description of PS 300 Control Unit in the *Document Set* is for systems with a larger cabinet and a GCP. The PS 390 has six basic circuit cards: JCP, Mass Memory (MM), Arithmetic Control Processor (ACP), Pipeline Subsystem (PLS), Frame Buffer and Bit–Slice Processor (FBL/BP), and Frame buffer and Video Controller (FBR/VC). The architecture for the PS 390 is shown in Figure 1.



JCP –	JOINT (GRAPHICS) CONTROL PROCESSOR	FBL –	FRAME BUFFER LEFT
MM –	MASS MEMORY (1– to 4–MBYTES)	BP –	BITSLICE PROCESSOR
ACP –	ARITHMETIC CONTROL PROCESSOR	FBR –	FRAME BUFFER RIGHT
PLS –	PIPELINE SUBSYSTEM	VC –	VIDEO CONTROLLER
		GPIO –	GENERAL PURPOSE INTERFACE OPTION

Figure 1. PS 390 Architectural Overview

The free–standing control unit of the PS 390 requires no clearance for operation, provided that site–specific heat dissipation requirements are met. It is mounted on casters for easy portability and to provide return air to the unit fan.

The control unit is approximately 53 cm (21 inches) wide, 71 cm (28 inches) deep, 67 (26.5 inches) high, and weighs 55 kg (120 pounds). The top holds over 250 pounds static weight; 180 pounds rolling load. (See Figure 2.)

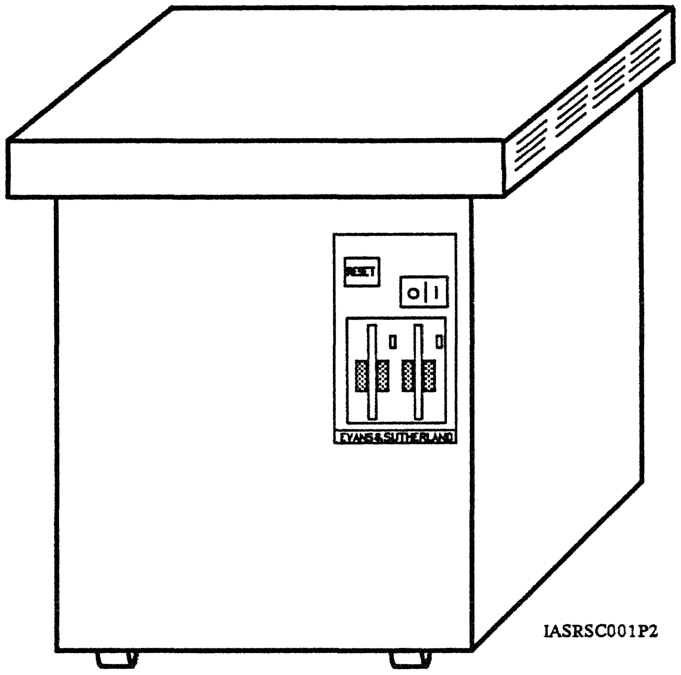


Figure 2. PS 390 Control Unit

There are two external controls on the PS 390 control unit. One is the ON/OFF circuit breaker switch, located at the top right of the front panel. This switch is recessed and surrounded by a protective frame. A RESET switch is located just left of the circuit breaker. The RESET switch allows the system to be reset instead of powered off during a system lock or reboot.

The PS 300 floppy disk drives are located at the front of the unit near the upper, right corner.

The PS 390 uses a double-sided, quad-density, 5-1/4 inch minifloppy diskette capable of storing 737,280 formatted data bytes on 160 tracks.

At the back of the control unit, above the power distribution panel, is the communications connector panel. See Figure 3. The panel is vertically aligned, with ports 0-5 from the top down. Connectors are externally accessible on the back of the control unit.

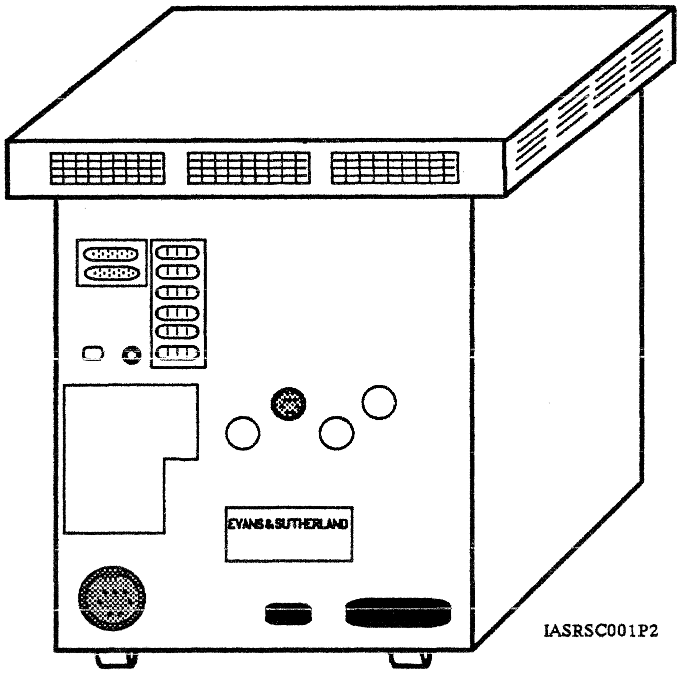


Figure 3. Back of Control Unit

The standard PS 390 control unit comes with the cards in place. The metal casing on the inside of the unit replaces the Faraday cage installed in some older cabinets.

The PS 390 is FCC Class A certified for emissions and will meet UL 478 and CSA 22.2 #154 safety standards.

The new Joint Control Processor (JCP) card consists of two (optionally three) sections: Control Processor, Mass Memory, and Interface section.

The control processor (CP) section is functionally similar to the old graphics control processor (GCP) and is based on a 68000 10 MHz microprocessor.

This differs from the GCP card documented in the Document Set in that:

- Local memory is increased from 256K to 512K.
- There is a local path to the JCP resident mass memory that is used instead of the Common bus path (GCP systems) thus providing faster access to mass memory from the 68000.
- Four usable asynchronous RS-232 ports are supported (compared to five on the GCP) which reside on the Communications Connector Panel. See Figure 4. Port 0 and Port 2 are physically present but not usable. This means that the DMR-11 interface is not available with reduced size cabinet systems nor is multi-user functionality.

The new port configuration for the PS 390 is as follows:

Port 1 is the host port.

Port 3 is the debug port, for diagnostic purposes.

Port 4 may be used for special interface applications, including an alternate diagnostic port.

Port 5 is used for the peripheral multiplexing box and therefore, is not available for your use.

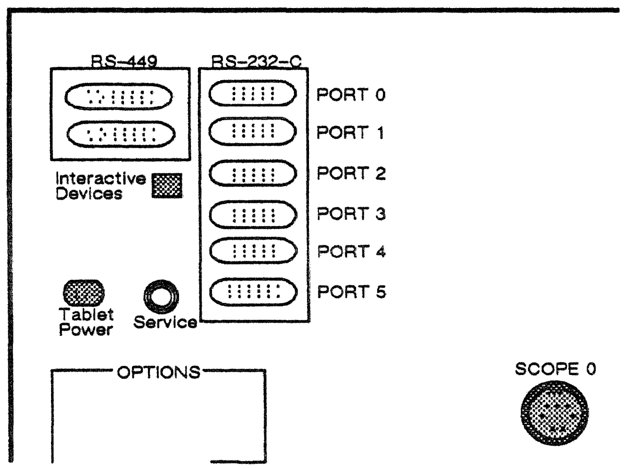


Figure 4. Port Configuration

- The Mass Memory section of the JCP card has one megabyte of memory with the option of a second megabyte available.
- The interface section of the JCP provides a location for the optional IBM 3278 interface. This option allows the PS 390 to communicate with an IBM 3274 control unit over a 56KB line. It is functionally equivalent to the PS 300 IBM 3278 GPIO card. Separate GPIO cards are available for high-speed communication interfaces other than IBM 3278.

2.2 Operating Specifications

Operating specifications for the PS 390 are as follows.

Grounding - The PS 390 scope should share a common ground with the control unit

Power Requirements - 115V Single Phase ±10% 47-63 Hz, 12 amp (max)
220V Single Phase: 7 amps (max) for the control unit

The following limitations are placed on AC power disturbances:

- A maximum of ±10% of nominal power for .1 seconds occurring no more than once every 10 seconds.
- Maximum harmonic content of 5% rms, no more than 3% rms for any single harmonic.
- Maximum impulse of 300V with rise time of .1 microseconds or slower, lasting no longer than 10 microseconds total duration.

Power Consumption - 1380 watts maximum

Heat Dissipation - 4710 BTUs/hour maximum

Operating Temperature - 65° to 80°F (18° to 27°C)

Relative Humidity - 20% to 80%

2.3 Multiplexing Box and Peripheral Connections

Peripherals for the PS 390 are connected to a multiplexing box contained in a three-inch pedestal that supports the raster scope. Mux boxes for either set of peripherals supported by PS 390 have the same operating instructions noted here. All peripheral connections for the mouse, function buttons, control dials, keyboard and tablet are clearly marked on the front panel of the mux box. Figure 5 shows the front view of the mux box.

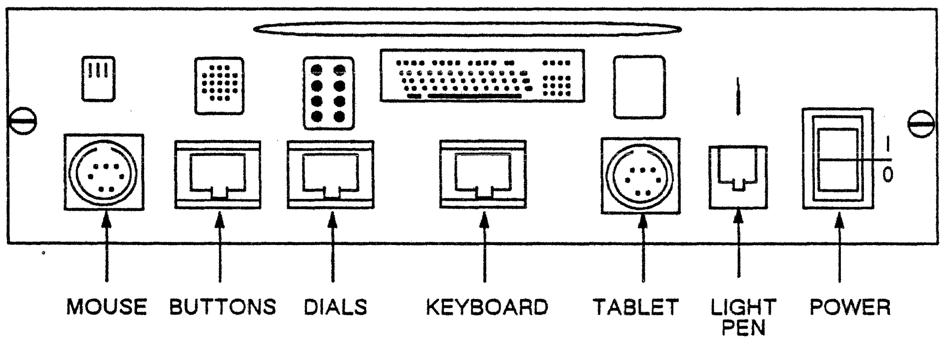


Figure 5. Front View of Multiplexing Box

The back panel of the multiplexing box has an RS232-C connection, three external power connections and two BNC connections marked LPICK and TPSW. The BNC connections are reserved for future use. All cables and connections are clearly marked. To maintain EMI integrity, the screws on the RS232-C shielded cable must be tightly turned on the connection. The rear panel of the mux box is shown in Figure 6.

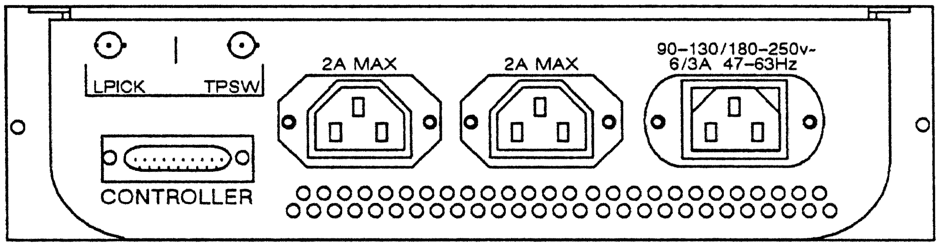


Figure 6. Rear Connections of Mux Box

Documentation on the new-style peripherals and multiplexing boxes is included as a separate manual with this release. Documentation for previous peripherals is contained in Volumes 1 and 5 of the *PS 300 Document Set*.

3. RUNTIME MODIFICATIONS AND NEW FEATURES

PS 390 runtime firmware supports new PS 390 functionality and existing PS 300 functionality. Assume that all functionality described in the *PS 300 Document Set* is correct and applicable to the PS 390 unless specifically noted as different in this and following sections.

As previously mentioned, the primary concern in developing the PS 390 runtime was maintaining user software compatibility with previous systems. Some incompatibilities exist because the reduced-size control unit does not support ports 0 and 2 for use with the DMR-11 interface or multi-user systems.

3.1 New **PS 390 Function**

A new initial function instance, PS390ENV, is provided. This function sets up display background color, and selects cursor and cursor color.

Input <1> is a trigger which accepts any data type to cause the function to run.

Input <2> is a constant which accepts a 3D vector (hue, saturation and intensity) to specify background color. The default background color is 0,0,0 (black). Saturation and Intensity must be in range of [0,1], otherwise an error message will be generated. Hue is in the range of [0,360]. For any value specified outside this range, multiples of 360 are added or subtracted to bring it into this range.

Input <3> is a constant which accepts an integer in the range [0,7] to specify the cursor color where

0 = black
1 = blue
2 = green
3 = cyan
4 = red
5 = magenta
6 = yellow
7 = white (default)

Any value outside this range generates an error.

Input <4> is a constant which accepts an integer to select the cursor.

0 = update rate cursor (default)
1 = system-defined refresh cursor

Input <5> accepts an integer to specify the video timing format, which is output from the video connection on the back of the PS 390 control unit.

0 = 1024 x 864 non-interlaced (default required by the PS 390 display)
2 = 1024 x 864 interlaced
3 = 640 x 484 interlaced (RS-170)

NOTE

When specifying the system-defined refresh rate cursor, you should leave the initial viewports HVP1$ and GVP0$ unchanged in order to have the (hardware) cursor work with picking.

3.2 Dynamic Viewport Considerations

Although the raster screen contains 1024 by 1024 addressable pixels, the actual displayable area on the raster screen is a rectangle, with pixel addresses going from 0 to 1023 in X and 0 to 863 in Y, where the physical pixel address 0,0 is in the lower left corner. A PS 300 viewport which spans (-1,1) in both vertical and horizontal directions maps onto the full 1024 x 1024 screen so that a rectangular portion along the lower edge of the viewport is not displayed. To avoid this situation, all viewports in the display structure are initially concatenated with a default viewport in the top display structure which maps to a square of 864 x 864.

The command

```
VPF1$ := Viewport Horizontal = -0.825:0.825 Vertical = -0.65:1 Intensity = 0:1
Then HVP1$;
```

in the boot-time configuration file accomplishes this.

If you want to override the default and use the entire displayable rectangular screen area, the following command can be entered:

```
Configure A;
VPF1$ := Viewport Horizontal = -1:1 Vertical = -0.65:1 Intensity = 0:1
```

```
Then HVP1$;
Finish Configuration;
```

This will cause all the subsequent VIEWPORT commands in the structure to be concatenated with this rectangular viewport. In doing so, however, your data must account for the non-square viewport.

To re-establish the default viewport, use either the commands

```
Configure A;
VPF1$ := Viewport Horizontal = -0.825:0.825 Vertical = -0.65:1 Intensity = 0:1
Then HVP1$;
Finish Configuration;
```

or

```
Screensave := F:Screensave;
```

Note that the initialize command does not restore the original viewport. Also, note that you cannot override the default viewport with the LOAD VIEWPORT command.

3.3 "Soft Labels" Function Network

Included on the distribution tape in the PS 390 Subdirectory is the "soft labels" ASCII file, which sets up a structure and network to use a normally unused portion of the screen to display function key and dial labels. This file can be incorporated in your SITE.DAT file if you have the new peripherals without LED labels. The labels appear on the left-hand side of the screen, with the square, default, graphics viewport shifted fully to the right. The displayed labels provide visual feedback to the user, but they are not pickable.

The structure and network requires no application software changes, except that the label, **flabel10**, no longer exists. (This is the 96 character label that goes across the entire LED area of the standard E&S keyboard with LEDs.) Figure 7 shows this soft labels area as it appears on the screen.

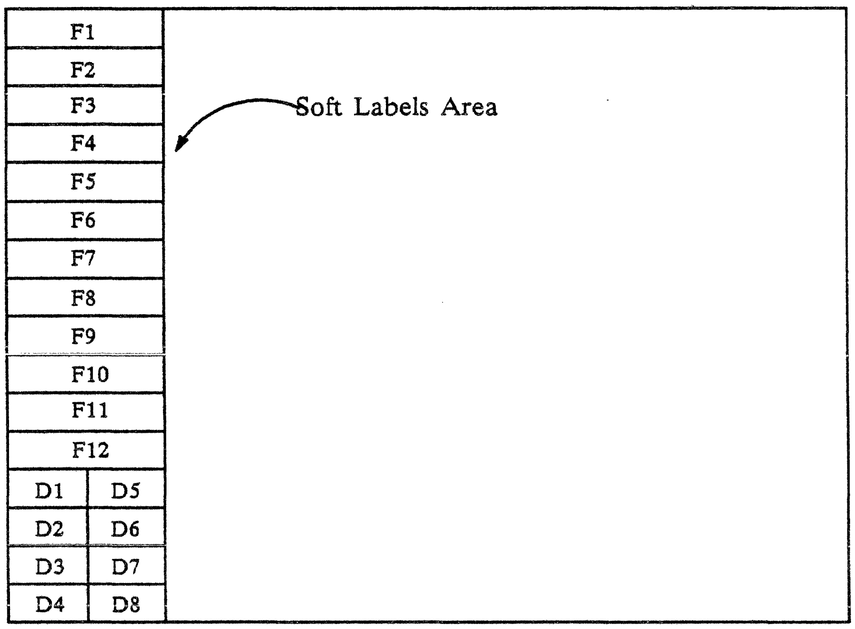


Figure 7. Screen Layout for the PS 390 with Soft Labels

3.4 Multiple GPIO Interfaces

The PS 390 runtime firmware supports up to two GPIO interfaces of differing types as well as asynchronous communications installed in the same system. The default configuration is asynchronous, but you have the ability to change your default to configure any interface when the system is booted. This is explained in the following section.

It is also possible to change the configuration without rebooting the PS 390 because the runtime determines which of the interfaces are in the system and initializes them all. This is achieved through runtime identification of up to two GPIOs at the first two addresses assigned to GPIO interface cards. (Refer to Send 'UNIBUS' command in 3.4.1 for an example of how to do this.) However, there are some limitations to the use of multiple GPIOs. First, there cannot be two of the same type GPIOs in the same system. Second, if the IBM 3278 option is included, then only one additional GPIO may be added. The 3278 GPIO running under previous PS 300 systems is not supported under PS 390. Table 1 shows the possible GPIO combinations.

Table 1. Possible GPIO Combinations

1st GPIO	2nd GPIO
IBM 3278 (enabled on JCP)	IBM 5080
	Parallel
	Ethernet/DECNET
IBM 5080	Parallel
	Ethernet/DECNET
Parallel	IBM 5080
	Ethernet/DECNET
Ethernet/DECNET	IBM 5080
	Parallel

3.4.1 Interface Configuration Files

The PS 390 runtime is distributed on two diskettes and contains more files than previous PS 300 runtime diskettes. This is to allow for the many different combinations of interfaces possible with the multiple GPIO operation.

When the PS 390 is booted, the system attempts to read the file, **INTFCFG.DAT**. If this file is not found, the system will boot with the default interface of Asynchronous, and display the message **INTFCFG.DAT NOT FOUND**. To boot with a default interface in addition to Asynchronous, the appropriate interface file must be renamed to INTFCFG.DAT. This can be done using the Diagnostic Disk Utility program described in Volume 5, Section 10 of the *Document Set*. For example,

```
Rename  ETHERNET.DAT  INTFCFG.DAT
```

would rename the default interface to Ethernet so that, at boot time, the communications interface protocol for Ethernet would be configured.

The following is a list of the file names on the diskette and which interface each file sets up.

ASYNC.DAT	Asynchronous communications
IBM3278.DAT	IBM 3278 communications
IBM5080.DAT	IBM 5080 communications
UNIBUS.DAT	Parallel interface communications
ETHERNET.DAT	Ethernet communications (for Ethernet or Decnet)

If your system hardware supports two interfaces, you can change the interface during a session without rebooting by sending the name of the interface file to input <1> of RDCFG$. For example, the following command,

```
Send 'UNIBUS' to <1>RDCFG$;
```

would change the communications protocol to the UNIBUS Parallel interface to allow parallel communications.

Table 2 shows the files contained on the PS 390 diskettes which are needed for a particular interface.

Table 2. Required Interface Files

PS 390 File Name	*Async*	*3278*	*5080*	*Unibus*	*Ethernet*
mmdd390J.EXS	✓	✓	✓	✓	✓
ACPCODE2.DAT	✓	✓	✓	✓	✓
ASYNC.DAT	✓				
CHARFONT.DAT	✓	✓	✓	✓	✓
CIRCLE.DAT			✓		
CONFIG.DAT	✓	✓	✓	✓	✓
DINTCODE.DAT					✓
EINTCODE.DAT					✓
ETHERNET.DAT					✓
FCNDICTY.DAT	✓	✓	✓	✓	✓
FCNTABLE.DAT	✓	✓	✓	✓	✓
FONT5080.DAT			✓		
GPIOCODE.DAT		✓			
HMSCODE.DAT	✓	✓	✓	✓	✓
HMSCOL.DAT	✓	✓	✓	✓	✓
HMSVEC.DAT	✓	✓	✓	✓	✓
IBM3278.DAT		✓			
IBM5080.DAT			✓		
IBMASCII.DAT	✓	✓	✓	✓	✓
IBMFONT.DAT	✓	✓	✓	✓	✓
IBMKEYBD.DAT	✓	✓	✓	✓	✓
INITACP.DAT	✓	✓	✓	✓	✓
INITGPIO.DAT		✓			
LINLUT.DAT	✓	✓	✓	✓	✓
LUT.DAT	✓	✓	✓	✓	✓
MSGLIST.DAT	✓	✓	✓	✓	✓
OVERLAY2.DAT	✓	✓	✓	✓	✓
PARSECODE.DAT	✓	✓	✓	✓	✓
PARSDICT.DAT	✓	✓	✓	✓	✓
PINTCODE.DAT				✓	
SINE.DAT			✓		
THULE.DAT	✓	✓	✓	✓	✓
UNIBUS.DAT				✓	

All of the interface files assume that the keyboard used is a VT100-style keyboard. A FALSE is sent to the keyboard handler (either IBMKBD or KBHANDLER) at the end of the file. To use the IBM-style keyboard, the command in the interface file must be changed to send TRUE to the keyboard handler. For example,

```
    Send True to <2>Kbhandler;
```

would accomplish this.

This also means that full VT100 support is provided with the IBM-style keyboard.

(Please note that the IBM keyboard is not supported with the initial release of the PS 390.)

3.4.2 Ethernet/DECNET Interface

The GPIO interface hardware for Ethernet and DECNET is the same. The only difference is the microcode that is loaded into the GPIO. Therefore, both microcode files are distributed on each disk. The runtime attempts to load a file named EINTCODE.DAT. Ethernet is the default on the disk. The file for the DECNET interface is DINTCODE.DAT. If your system supports a DECNET interface, you must rename DINTCODE.DAT to EINTCODE.DAT to load the DECNET microcode into the GPIO. This can be accomplished by using the Diagnostic Utility program.

NOTE

As documented in the *Customer Installation and User Manual PS 300 Ethernet Interface*, you must send the assigned Ethernet address to the PS 300. The command to do this for the PS 390 is

```
    Send 'address' to <1>ei_o1$;
```

Please refer to that manual for instructions on doing this.

3.5 Crash Dump File

Another of the new features of the runtime is the writing of a Crash Dump file to the diskette in drive 0 when a system crash occurs. This file is always named Crash.dat;1 and occupies only 1 block on the diskette.

If the file already exists it will be overwritten by the new crash information. If the file doesn't exist, it will be created. If there is insufficient room on the disk for the file, no crash dump file will be written.

The file consists of the 8 Data, the 8 Address registers, system version, system type, program counter, error type, error number, 59 32-bit stack entries, and the 68000 status register. Figure 8 shows the structure of the data in the crash file. Appendix A gives more information on some of these values.

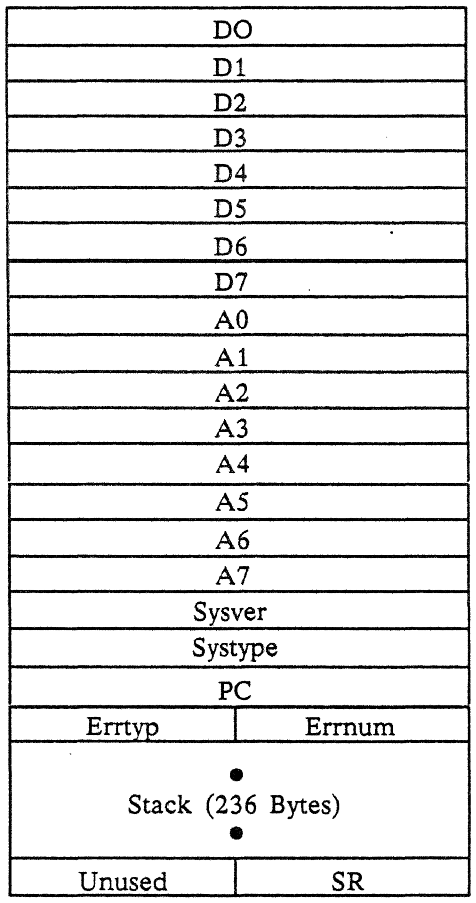


Figure 8. Data in Crash File

Appendix A also gives an example of a host PASCAL program that reads back the Crash.dat file from a PS 300. This program will read the Crash.dat file from a PS 300 and display the information in a format similar to the debug port on the PS 300. This information can be helpful in determining the cause of a crash.

The READDISK function has an added constant input <2> which accepts a boolean. If there is a true on input <2> after the file specified on input <1> is read, the file is deleted.

One possible use of this function is that an application program on the host could read and maintain crash file information. For example, a host program could have a start up procedure that checks to see if a crash file exists and then logs it in a host file. By reading and then immediately deleting this file, the program prevents the logging of crash files that were already recorded. The existence of a crash file would indicate that a crash had occurred since the last time the host program was run.

3.6 Additions to F:PICK

The PS 390 pick function, F:PICK, has three additional inputs. Input <4> is a real number between 0 and 1 that defines the pick window half size for the ACP pass of the pick . This is different from the size set by the SET_PICKing_LOCation operation node. The Line Generator or the Frame Buffer uses the operation node to determine if a pick has occurred, while the ACP uses input <4> to do the actual pick pass on the data.

Input <5> is an integer specifying pick pass retries. Since it is possible that the ACP will not find the picked data during a pick pass, input <5> indicates the number of times to add the window half-size increment on input <6> and try another pick pass.

Input <6> is a real number between 0 and 1 which specifies the amount to increase the pick window half size on each retry of the pick pass.

The defaults for each input are:

Input <4>	6.8359E-3
Input <5>	4
Input <6>	6.8359E-3

3.7 UWF Runtime Code Modification

The stack allocation scheme for User-written functions (UWF) has been changed. The UWF stack is now allocated when functions are downloaded rather than when they execute. As each function is processed by the SREC_GATHER function, the stack size requested is checked against the size of the currently allocated stack. If the requested stack size is the same or smaller, no action is taken. If the requested stack size is larger than the currently allocated stack, the current stack is disposed and a new one is allocated. All UWFs therefore use the same stack area. This allocation scheme eliminates the time previously required to allocate and deallocate a stack block on each execution of a UWF. Note that when UWFs are used, the one with the largest stack request should be loaded first.

4. PS 390 EXCEPTIONS

PS 390 functionality is the same as described in existing PS 300 documentation with the exception of the following.

There is no support for Port 0 and Port 2 of the control unit, DMR-11 interface, multi-user, or any scope other than scope 0.

The default viewport of the PS 390 is 864x864 centered on the raster display. You can change this by using the commands described in Section 3.2, but you cannot modify the this default viewport using the Load Viewport command.

Local hardcopy is not supported. Host hardcopy is available through the Writeback feature included with this release.

Reading back of pixel data from the PS 390 to the host is not supported nor is the loading of user-defined color lookup tables or the display of anti-aliased objects in overlay mode. This functionality is planned for future releases.

The lightpen is not supported.

There are no vector-normalized vectors; all ASCII and GSR vector list commands which do not specify block-normalized vectors will create 32-bit block-normalized vectors internally in the PS 390. (No modifications to ASCII commands or GSR routines are required.)

There is no "per vector" intensity specification available.

There is no color blending (color by vector) available.

Zero length vectors cannot be picked.

There is no allowance for the display of transformed data (data output by F:XFORMDATA). However, a limited form of access to the data generated by the F:XFORMDATA function has been provided to allow certain user-written functions (and the CPK modeling firmware) to perform properly. Please note the following restrictions on the use of transformed data on the PS 390:

- F:XFORMDATA outputs a non-displayable data type (vector-normalized vector list).
- A single-precision vector list is generated by F:XFORMDATA.
- Only three-dimensional data can be transformed.

F:XFORMDATA can still be connected to F:LIST to enable the host to read the transformed data retrieved from the PS 390.

Existing PS 300 applications that create nodes with functionality not yet supported by the PS 390 will be treated as no operation nodes.

APPENDIX A

Crash Dump Information

The System Version is a number generated indicating the date the runtime was created. For example, a value of 111486 means the system was created on Nov. 14, 1986.

The system type is a three digit number indicating the type of system that is being used. The first digit on the left is a 1 for GCP, 2 is reserved, 3 for JCP. The second digit, 1, is reserved, 2 is for 320, 3 is for 330, 4 is for 340, 5 is for 350 and 6 is for 350/340. The last digit is 0 for Async or any JCP, 1 for IBM 3278, 2 for Parallel, 3 for IBM 3250/5080 and 4 for Ethernet/DECNET. Digit 3 will always be 0 for JCP systems. For example a value of 350 indicates JCP 350.

Error Types/Error Numbers

There are three crash error types in the PS 300. Each type has a set of error numbers associated with the type. The three types are:

1. System Errors
2. Traps
3. Exceptions

The following is the list of errors for each type.

Type 1 - System Errors

1	Track number out of range
2	Disk drive not ready
3	Disk remains busy after a seek
4	Block number out of range

6	Lost data during read
7	Record not found during read
8	Data CRC error during read
9	ID CRC error during read
B	Lost data during write
C	Record not found during write
D	Data CRC error during write
E	ID CRC error during write
F	Write fault
10	Disk is write protected
11	Lost data during format
12	Write fault during format
14	Disk drive number out of range
15	Seek error
16	Drive not ready during read
17	Drive not ready during write
18	Disk not at track Ø after restore command
19	Disk busy after restore command
1A	Track number out of range during format
1B	Drive not ready during format
1C	Disk write protected during format
1D	Time out during read
1E	Time out during write
1F	Time out during format
64	Wait maybe called with nil argument
65	Wait maybe called with a non-function
66	Wait maybe, already a function waiting
67	Wait maybe, parameter function waiting elsewhere
68	Q ship to an unrecognized Namedentity
69	Msgcopy, Message type shouldn't be copied
6A	Msgcopy, Msg type Has structure, unknown to Msgcopy
6B	Send, 'Me' = nil
6C	Send, 'Me' not a function instance
6D	Send, No such output port for this function
6E	Rem_conn/Add_conn, A1 = nil
6F	Add_conn, A2 = nil
70	Findqueue, Named item = nil
71	Findqueue, illegal queue number (queue no. < Ø or queue no. > no. of inputs for function)
72	Allinpwait, Nmin > Nmax
73	Allinpwait, Nmin < 1
74	Tmessage, Waiting and n = Ø
75	Cmessage, Waiting and n = Ø
76	Lookmessage, Waiting and n = Ø
77	Allinputs, Nmin > Nmax

78	Allinputs, Nmin < 1
79	Fcnnotwait, Me = nil
7A	Findqueue, found a nil queue!
7B	Waitnextinput, n = Ø
7C	Anyoutputs, Me = nil
7D	Anyoutputs, illegal outset number
7E	Anyoutputs, no outset where there should be
7F	Fdispatch, function failed to re-queue after running
80	Text_text, B1 < Ø
81	Char_text, b < Ø
85	Error during disk read
8D	Initial structure not correct
8E	AnnounceUpdate List tail = nil;head < > nil
8F	FormatUpdate Somebody's sleeping in my bed
90	FormatUpdate Ready Head not nil but Tail is
91	Bad code file -- illegal Op
92	ByteIndex Invalid Acpdata type
93	FormatUpdate, PASCAL Head not nil but Tail is
94	Vec_size, Invalid Acpdata type
95	KillUpdate, Updfetch was < Ø
96	KillUpdate, Some one was sleeping in my bed
97	Vec_bias, Invalid Acpdata type
99	CntCapacity, Invalid Acpdata type
9C	Unknown brand of Namedentity
9D	Hasstructure knows something I don't
9E	Amuhead not a Qalphapair
A1	AppendVector, Invalid Acpdata type
A3	Nomemsched, Bad .Status for a fcn
A9	Bad update list on ACP time-out
AA	ACP Timeout during initialization
AB	Crashprepare, Name CRASH$ has not been defined
AC	DecUpdsync, C_header ^ .Updsync < Ø
AD	FormatUpdate, Someone waiting in C_header ^ .Updswait already
AF	Someone else waiting in C_header ^ .Killer already
BO	Non-nil Qwait of a dying function
B3	Microcode won't fit into ACP
B4	Implementation limit on delta waits (2**31)
B8	detected internal inconsistency
B9	detected error (passed a bad parameter)
BA	diskette's parsecode table inconsistent with parser
BD	Bad boundary on binary data xfer
BF	default Devsts contains errors
C0	Inwait, f is already waiting or not a function
C1	Outwait, f is already waiting or not a function
C2	ECO Level of GCP does not support 56K Baud Line

C3	Port 1 Configuration is invalid for 56K Baud Line Support
C9	User generic function stack overflow
CA	Ug_run_cnt has become negative
CB	User generic function has bad alpha (on private queue)
CC	Bad format of MSGLIST .DAT detected
CD	MSGLIST (or code using it) has probably been corrupted
CF	Apparent datastructure incompatibility
D0	Bad MemOKindex detected
D1	routine passed bad parm (e.g., a nil ptr)
D2	Lines to IBM system not active
D3	Floppy disk file INITGPIO.DAT; not found or unable to read
D4	Floppy disk file GPIOCODE.DAT; not found or unable to read
D5	Floppy disk file IBMFONT.DAT; not found or unable to read
D6	Floppy disk file IBMKEYBD.DAT; not found or unable to read
D7	Floppy disk file IBMASCII.DAT; not found or unable to read
D8	IBM GPIO timeout
D9	No. of minimum inputs is negative
DA	No. of maximum inputs < No. of minimum inputs
DB	No. of maximum inputs > # inputs for function
DC	Sendlist detected a bad list
DE	Sendmess: message to be sent is NIL
DF	Caller did not have a lock set already
E0	Curfcn in improper state to call Getinputs
E1	Cleanin, Curfcn in improper state to call Cleaninp (e.g., have you first called Getinputs?)
E2	Somebody remembered a forgotten non-fcninstance
E5	Alpha not already locked by caller
E6	Confusion in discarding bad message
E7	Lock not already set by caller
E8	Probable multiple master GCPs
E9	RemOne, Curfcn does not have that many inputs
EA	RemOne, Message to be deleted and message pointed to by Curinputs is not the same
EB	Lock not already set in Gatheraupdate call
ED	Get2locks detected lock already set
EE	Error in semantic routine for polygon vertex
EF	Destination Alpha was not already locked
F0	Parent not already locked in add/remove from set
F1	Child not already locked in add to set
F3	Alpha not already locked in Gpseudoaupdate
F6	Confusion about locks or decausages
F7	Unknown tap reason
F8	Unanticipated state at which to see shoulder tap
F9	Illegal number of inputs
FC	No existing DCB found for this user

FD	Timeout, Message on input 1 disappeared before fcn could get it
FE	Error while initializing disk drive
FF	Error while reading disk header
100	Error while reading disk directory
101	THULE.DAT not found on disk
102	Error while reading THULE.DAT
103	Curfcn was not active at entry
104	Viewport not in structure
105	Real_simple, number of digits requested out of range (n < 1 or n > 9)
106	Getnextone, illegal queue specified
107	Getnextone, msg on head of queue and specified by Curinput do not agree
108	Getnextone, no message on queue, but Curinput < > NIL
109	ContBlock, nil block
10A	Timeout when waiting for all on-line GCPs
10B	Rehash only works first time, only time now.
10C	No processor has right to issue this tap
10D	GetVector, Not an Acpdata block
10E	GetVector, Not a vector Acpdata block
10F	Invalid qpacket received
110	Tolerance on FCnearzero is absurd
111	set construct of father has no dummy control block
112	function code has to be of type CI to have elements included and removed
113	ShadeEnviron node encountered in non PS 340

Type 2 - Traps

0	No mass memory on line, or too little to come up
1	More OKINTs than NOINTs or > 128 NOINTs
2	Free storage block size bad (on request or in free list)
3	Attempt to Activate a non-function (or nil) or bad software detected during startup (most commonly, incompatible datastru.sa detected but perhaps invalid startup routine sequencing (if someone has been mucking around with it))
4	NEW call failed to find memory, within NOMEMSCHED
5	Attempt to queue where a function is already waiting
6	Systemerror(n)
7	Badfcode(Fcn)
8	Mass Memory Error Interrupt
9	Utility Routine not included in this linked system
A	Probable multiple DISPOSE of the same block
B	Block exponent not big enough
C	Attempt to divide with a divisor which is too small in FixLongDivide(twice the dividend must be less than the divisor)
D	(Used by Motorola PASCAL)

Type 3 – Exceptions

0	Reset: Initial SSP
1	Reset: Initial PC
2	Bus Error (i.e. attempt to address nonexistent location in memory)
3	Address Error (i.e. attempt to access memory incorrectly, for example an instruction not starting on a word boundary).
4	Illegal instruction
5	Zero Divide
6	CHK Instruction
7	TRAPV Instruction
8	Privilege violation
9	Trace
10	Line 1010 Emulator
11	Line 1111 Emulator
24	Spurious interrupt

Crash Dump Program

Following is an example of a Pascal host program that writes the information from the PS 300 crash file into a host file.

```
PROGRAM CRASH (Input,Output,Outfile);

CONST
        %INCLUDE 'PROCONST.PAS/NOLIST'

TYPE
        %INCLUDE 'PROTYPES.PAS/NOLIST'

        cheat_4 = RECORD

                CASE Boolean OF
                        TRUE : (i : Integer);
                        FALSE : (c : Array[1..4] OF CHAR)

                END;

        cheat_2 = RECORD

                CASE Boolean OF
                        TRUE : (i : [WORD] 0..1024);
                        FALSE : (c : Array[1..2] OF CHAR)

                END;

        Buffer = RECORD
                CASE Boolean OF
                        TRUE : (b : P_VaryBuftype);
                        FALSE : ({ Length of P_VaryBuftype is in Dummy}
                                Dummy           :[WORD] 0..1024;
                                Dreg            :Array[0..7] of Cheat_4;
                                Areg            :Array[0..7] of Cheat_4;
                                SVer            :Cheat_4;
                                Stype           :Cheat_4;
                                PC              : Cheat_4;
                                Errtyp          : Cheat_2;
                                Errnum          : Cheat_2;
                                Stack           : Array[1..59] of Cheat_4;
                                Not_Used        : Cheat_2;
                                SR              : Cheat_2)

            END;
```

```
VAR
        Devtyp: Integer;
        Inbuff : P_VaryBuftype;
        OutBuff: Buffer;
        Found : BOOLEAN;
        Outfile : text;

%INCLUDE 'PROEXTRN.PAS/NOLIST'

%INCLUDE 'VAXERRHAN.PAS/NOLIST'

PROCEDURE Init_ps300;

         FUNCTIONAL DESCRIPTION:

             Initialize the comm link to the PS 300


         }


         VAR
             a, Modify : P_Varyingtype;

         BEGIN
         Write('Enter Type of Interface (1=Async, 2=Ethernet, 3=Parallel):');
         Readln( Devtyp );
         Write('Enter Device name :');
         Readln( a ); CASE Devtyp OF
             1 :
               Modify := 'LOGDEVNAM=' + a + '/PHYDEVTYP=ASYNC';
             2 :
               Modify := 'LOGDEVNAM=' + a + '/PHYDEVTYP=ETHERNET';
             3 :
               Modify := 'LOGDEVNAM=' + a + '/PHYDEVTYP=PARALLEL'
             OTHERWISE

             END;
         PAttach( Modify, PI_Error_handler)
         END;
```

```
PROCEDURE Trigger_read;

        FUNCTIONAL DESCRIPTION:

             Create instance of function network to retrieve CRASH.DAT file from disk. The
             network will convert the data block to six-bit format and break it into packets of 72
             bytes which will be put on host_message.


        }

        VAR

            a : CHAR;

PROCEDURE BREAKUP;
{ Code generated by Network Editor 1.08 }
{ This function network takes an incoming qpacket and breaks it }
{ into smaller packets to be sent over a terminal line since  }
{ most terminal handlers have some limit to the input length  }
{ BREAKUP }
BEGIN
{ Frame1: }
    PFnInstN ('Break_sync', 'SYNC', 2,
        PI_Error_handler);
    PFnInst ('Break_route', 'BROUTEC',
        PI_Error_handler);
    PFnInst ('Add_constant', 'CONSTANT',
        PI_Error_handler);
    PFnInst ('Break_add', 'ADDC',
        PI_Error_handler);
    PFnInst ('Breakup', 'TAKE_STRING',
        PI_Error_handler);
    PFnInst ('In_length', 'LENGTH_STRING',
        PI_Error_handler);
    PFnInst ('Len_compare', 'GTC',
        PI_Error_handler);
    PFnInst ('Route_string', 'BROUTE',
        PI_Error_handler);
    PFnInst ('Route_start', 'BROUTE',
        PI_Error_handler);
    PFnInst ('cvt', 'CVT8TO6',
        PI_Error_handler);
    PFnInst ('rd', 'READDISK',
        PI_Error_handler);
```

```
PFnInst ('prnt', 'PRINT',
    PI_Error_handler);
PFnInst ('Breakup_in3', 'CONSTANT',
    PI_Error_handler);
PConnect ('Break_sync', 1, 1, 'Breakup',
    PI_Error_handler);
PConnect ('Break_sync', 1, 2, 'Break_route',
    PI_Error_handler);
PConnect ('Break_sync', 2, 2, 'Breakup',
    PI_Error_handler);
PConnect ('Break_sync', 2, 2, 'Break_sync',
    PI_Error_handler);
PConnect ('Break_sync', 2, 2, 'Break_add',
    PI_Error_handler);
PConnect ('Break_route', 1, 1, 'Add_constant',
    PI_Error_handler);
PConnect ('Break_route', 1, 2, 'Route_string',
    PI_Error_handler);
PConnect ('Add_constant', 1, 1, 'Break_add',
    PI_Error_handler);
PConnect ('Break_add', 1, 2, 'Break_add',
    PI_Error_handler);
PConnect ('Break_add', 1, 2, 'Route_start',
    PI_Error_handler);
PConnect ('Break_add', 1, 1, 'Len_compare',
    PI_Error_handler);
PConnect ('Breakup', 1, 1, 'cvt',
    PI_Error_handler);
PConnect ('Breakup', 2, 1, 'Break_route',
    PI_Error_handler);
PConnect ('Breakup', 2, 1, 'Breakup_in3',
    PI_Error_handler);
PConnect ('In_length', 1, 2, 'Len_compare',
     PI_Error_handler);
PConnect ('Len_compare', 1, 1, 'Route_string',
    PI_Error_handler);
PConnect ('Len_compare', 1, 1, 'Route_start',
    PI_Error_handler);
PConnect ('Route_string', 2, 1, 'Breakup',
    PI_Error_handler);
PConnect ('Route_start', 2, 2, 'Breakup',
    PI_Error_handler);
PConnect ('cvt', 1, 1, 'host_message',
    PI_Error_handler);
PConnect ('rd', 1, 1, 'Break_sync',
        PI_Error_handler);
    PConnect ('rd', 1, 1, 'In_length',
        PI_Error_handler);
    PConnect ('rd', 2, 1, 'prnt',
        PI_Error_handler);
    PConnect ('prnt', 1, 1, 'host_message',
        PI_Error_handler);
    PConnect ('Breakup_in3', 1, 3, 'Breakup',
        PI_Error_handler);
    PSndStr(CHR(36), 2, 'cvt',
        PI_Error_handler);
    PSndFix (48, 3, 'Breakup',
        PI_Error_handler);
    PSndFix (48, 2, 'Breakup_in3',
        PI_Error_handler);
    PSndFix (48, 2, 'Add_constant',
        'PI_Error_handler);
    PSndFix (1, 2, 'Break_sync',
        PI_Error_handler);
    PPutPars('Set priority of prnt to 9; ',
        PI_Error_handler);
END;

BEGIN
IF Devtyp = 1
THEN
    Breakup
ELSE
    BEGIN
    PFnInst ('rd', 'READDISK',
        PI_Error_handler);
    PFnInst ('prnt', 'PRINT',
        PI_Error_handler);
    PConnect ('rd', 2, 1, 'prnt',
        PI_Error_handler);
    PConnect ('prnt', 1, 1, 'host_message',
        PI_Error_handler);
    PConnect ('rd', 1, 1, 'host_message',
        PI_Error_handler);
    PPutPars('Set priority of prnt to 9; ',
        PI_Error_handler);
    END;
Write(' Do you want to delete CRASH.DAT after reading?');
Readln( a );
IF (a = 'Y') OR (a = 'y')
```

```
THEN
    Psndbool( TRUE, 2, 'rd', PI_Error_handler)
ELSE
    Psndbool( FALSE, 2, 'rd', PI_Error_handler);
Psndstr( 'CRASH', 1, 'rd', PI_Error_handler)
PPurge( PI_Error_handler );
END;

PROCEDURE Get_data_block;

    {

    FUNCTIONAL DESCRIPTION:

            Read in data from PS 300, convert to 8 bit and put in buffer

    {
    VAR
        i,j,Temp : Integer;
        Done : BOOLEAN;

PROCEDURE Cvt_6_8
        (Inblock : P_VaryBuftype;
        VAR Outblock : P_VaryBuftype;
        Factor : Integer);

    VAR
        w : cheat_4;
        c_out,cycle_count,il,tc : INTEGER;
        First : BOOLEAN;

    BEGIN
    i := 1;
    First := TRUE;
    Cycle_count := 1;
    c_out := 4;
    WHILE i <= LENGTH(Inblock) DO
        BEGIN
        tc := ORD(Inblock[i]) - Factor;
         IF First
        THEN
            IF tc < 0
            THEN
                c_out := 4 + tc
            ELSE
            BEGIN
            First := FALSE;
            w.i := tc;
            cycle_count := SUCC(cycle_count)
            END


        ELSE
            BEGIN
            w.i := w.i * 64;
            w.i := w.i + tc;
            cycle_count := SUCC(cycle_count)
            END;
        IF cycle_count > 6
        THEN
            BEGIN
            FOR il := 1 TO c_out DO
                Outblock := Outblock + w.c[il];
            cycle_count := 1;
            First := TRUE
            END;

        i := SUCC(i);
        END;
    END;


BEGIN
Done := FALSE;
Found := TRUE;
WHILE NOT Done DO
    BEGIN
    Pgetwait( Inbuff, PI_Error_handler);
    IF Inbuff = '''TRUE '''
    THEN
        Done := TRUE
    ELSE
        IF Inbuff = '''FALSE'''
        THEN
            BEGIN
            Done := TRUE;
            Found := FALSE
            END
        ELSE
            IF Devtyp = 1
            THEN
                Cvt_6_8( Inbuff, Outbuff.b, 36)
            ELSE
                FOR i := 1 TO 80 DO
                    FOR j := 4 DOWNTO 1 DO
                        Outbuff.b := Outbuff.b + Inbuff[(i-1)*4 + j];
    END;

{ It is necessary to reverse Errnum with Errtyp }
{ and Not_Used with SR }
    IF Found
    THEN
        WITH Outbuff DO
            BEGIN
            Temp := Errnum.i;
            Errnum.i := Errtyp.i;
            Errtyp.i := Temp;
            Temp := Not_Used.i;
            Not_Used.i := SR.i;
            SR.i := Temp;
            END;


    END;
```

```
PROCEDURE Display_crash;

    {

    FUNCTIONAL DESCRIPTION:

        Display Crash info on terminal

    {
    PROCEDURE Dumpit;

        VAR
            SP,j,k,sloc,cloc : INTEGER;
             tc : CHAR;
            Sline : PACKED ARRAY [1..15,1..16] OF
                    CHAR;


        BEGIN
        Rewrite(Outfile);
        WITH Outbuff DO
        BEGIN
        Writeln(Outfile);
        Write(Outfile,' PC=',HEX( PC.i, 8, 8 ));
        Write(Outfile,' SR=',HEX( SR.i, 4, 4 ));
        Write(Outfile,' STYPE=',Stype.i:3);
        Write(Outfile,' SVER=',Sver.i:6);
        Write(Outfile,' ETYPE=',HEX( Errtyp.i, 4, 4));
        Write(Outfile,' ENUM=',HEX( Errnum.i, 4, 4));
        Writeln(Outfile); Write(Outfile,'D0-D7=');
        FOR j := 0 TO 7 DO
            Write(Outfile,' ',HEX( Dreg[j].i, 8, 8));
        Writeln(Outfile);
        Write(Outfile,'A0-A7=');
        FOR j := 0 TO 7 DO
            Write(Outfile,' ',HEX( Areg[j].i, 8, 8));
        Writeln(Outfile);
        Writeln(Outfile);
        Writeln(Outfile,'STACK='); SP := Areg[7].i + 14;
        FOR j := 1 TO 15 DO
            BEGIN
            Sloc := (j-1) * 4 + 1;
            Cloc := 4;
            FOR k := 1 TO 16 DO
                BEGIN
    IF sloc < 60
    THEN
        BEGIN
        tc:= Stack[sloc].c[cloc];
        IF tc > CHR(127)
        THEN
            tc:= CHR(ORD(tc) - 128);
        IF (tc < CHR(32)) OR
            (tc = CHR(127))
        THEN
            Sline[j,k] := '.'
        ELSE
            Sline[j,k] := tc
        END
    ELSE
        Sline[j,k] := '.';
    Cloc := Cloc - 1;
    IF Cloc = 0
    THEN
        BEGIN
        Cloc := 4;
        Sloc := Sloc + 1
        END;
    END;
Write(Outfile,HEX( SP, 8, 8),' ');
Sloc := (j-1) * 4 + 1;
Cloc := 4;
FOR k := 0 TO 15 DO
    BEGIN
    IF sloc < 60
    THEN
        Write(Outfile,' ',HEX( ORD(Stack[sloc].c[cloc]), 2, 2))
    ELSE
        Write(Outfile,' 00');
    Cloc := Cloc - 1;
    IF Cloc = 0
    THEN
        BEGIN
        Cloc := 4;
        Sloc := Sloc + 1
        END;
    END;
Write(Outfile,'  ');
FOR k := 1 TO 16 DO
    Write(Outfile,Sline[j,k]);
```

```
            Writeln(Outfile);
            SP := SP + 16
            END
        END
     END;


    BEGIN
    IF Found
    THEN
        Dumpit
    ELSE
        Writeln(' Crash file not found ')
    END;


BEGIN
Init_ps300;
Trigger_read;
Get_data_block;
Display_crash;
PDetach( PI_Error_handler);
END.
```

PART II

Change Pages And Previous Graphics Firmware Release Notes

A consolidation of Versions A1.V02 and A2.V01 of the *PS 300 Graphics Firmware Release Notes* are included in this package. Current customers should already have this information. Also included are change pages specific to PS 390 functionality.

The following commands and functionality have been added since the publication of the *Document Set.* The new commands have been formatted as supplement pages for the *PS 300 Command Summary*. The list below gives the new commands and a brief description.

Load Viewport	Loads a viewport and overrides the previous viewport (can not be used to modify default viewport).
Set Blinking ON/OFF	Creates blinking nodes to specify whether blinking is enabled in the specified structure.
Set Blink Rate	Specifies the blink rate.
Set Line_Texture	Specifies pattern for hardware texturing of displayed lines.
Writeback	Enables writeback for the data structure below the writeback node.
Rawblock	Allocates memory that can be directly managed by a user–written function, or the Parallel or Ethernet Interfaces.

Version A1.V02 - March 1985

DOCUMENTATION INFORMATION FOR ALL USERS

Important corrections to errors in the PS 300 Document Set are provided on the following pages. Please note these changes in your document set. New pages for previously undocumented functions are included here.

Several documents have been changed. The documents and the changes are summarized below. If you would like to have the newest version of any of these documents, please contact your E&S Account Executive.

- **User-Written Functions:** revised to correct errors in the document and provide templates with more complete instructions, as well as more information on writing various types of functions.

NOTE

A2.V02 - This manual has been completely revised and included in the PS 300 Advanved Programming manual that has been provided to you as a seperate document for the A2.V02 release.

- **NETEDIT:** revised to support the new version of NETEDIT.
- **Introduction to Data Driven Programming Methodology:** notes have been added to this document to clarify misleading information.
- **PS 300 Application Notes:** new Notes have been added.

Version A1.V02 - March 1985

Information for PS 300/IBM 3278 Interface Users

Enhancements and New Features in the PS300/IBM 3278 Firmware

- The PS 300/IBM 3278 Terminal Emulator Setup mode now includes keys that will inhibit the display of the cursor, the PS 300 indicator characters, and the host indicator characters. Inhibition of these screen characters is accessed by entering Setup mode, (ALT/GRAPH or ALT/SETUP on the IBM 3278-style keyboard, SETUP on the VT100-style keyboards) and toggling the appropriate keys.

  Once in Setup (shown by the display of the PS 300 indicator character 'S' on the bottom line of the screen), the following new Setup features are available:

FUNCTION KEY	FEATURE
F6	Toggles the display of the PS 300 characters. Default is the display of the characters.
F7	Toggles the display of the host indicator characters. Default is the display of the characters.
F8	Toggles the display of the cursor. Default is display of the cursor.

Function keys F9 and F10 are used in conjunction with the PS 300/IBM 3250 Interface. Information on the use of these keys is available in the PS 300/IBM 3250 Interface User's Manual.

The adjustments made in Setup can be entered as PS 300 commands in the SITE.DAT file to set the appropriate characteristics at boot time.

The list below shows the characters that should be entered into the SITE.DAT file for each new feature.

For VT100-style keyboards, the appropriate character(s) must be inserted between a '↑Vo ↑Vo' header and trailer sequence. ↑Vo is a CTRL V lowercase "o" sequence:

FEATURE	CHARACTERS TO BE ENTERED INTO SITE.DAT
Set/Reset Local Indicators	SEND '↑Vo↑Vf↑Vo' TO <1>IBMKBD1;
Set/Reset Host Indicators	SEND '↑Vo↑Vg↑Vo' TO <1>IBMKBD1;
Set/Reset Cursor	SEND '↑Vo↑Vh↑Vo' TO <1>IBMKBD1;
Set 3250 Mode	SEND '↑Vo↑Vi↑Vo' TO <1>IBMKBD1;
Set PS300 Mode	SEND '↑Vo↑Vj↑Vo' TO <1>IBMKBD1;

For IBM-style keyboards, the appropriate characters must be inserted between a CHAR(130)&CHAR(n)&CHAR(130) sequence, where &CHAR(n) is the character sequence(s) for the feature:

FEATURE	CHARACTERS TO BE ENTERED INTO SITE.DAT
Set/Reset Local Indicators	SEND CHAR(130)&CHAR(150)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Host Indicators	SEND CHAR(130)&CHAR(151)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Cursor	SEND CHAR(130)&CHAR(152)&CHAR(130) TO <1>IBMKBD1;
Set 3250 Mode	SEND CHAR(130)&CHAR(153)&CHAR(130) TO <1>IBMKBD1;
Set PS300 Mode	SEND CHAR(130)&CHAR(154)&CHAR(130) TO <1>IBMKBD1;

- A kit containing keycap replacements for the 3278-style keyboard accompanies this release. The configuration of the keyboard has changed with the A1.V02 Firmware to support additional keys required by some IBM applications. The keyboard reconfiguration and keycap replacements are as follows:

  1. The keys designated for use by IBM applications are the old GRAPH and TERM keys on the left-hand keypad of the keyboard. These keycaps will be replaced by blank keycaps and have no PS 300 application.

  2. The old SETUP and TEST/NORM keys on the left-hand keypad will become dual-purpose keys. The new keycap for the SETUP key will read GRAPH on the top and SETUP on the front of the key. To access Setup mode, the key must be pressed in conjunction with the ALT key on the keyboard.

  The new keycap for the TEST/NORM key will read TERM on the top and TEST/NORM on the front. The terminal display will be toggled on and off by pressing the key. To access TEST/NORM, the key must be pressed in conjunction with the ALT key on the keyboard.

  The keycap exchange will be made by the user. Additional instructions for changing the keycaps are included in the kit.

Version A1.V02 - March 1985

- Two system functions (F:IBM_KEYBOARD and F:IBM_SETUP) have been modified to support the new PS 300/IBM 3250 Interface. The modifications made to these functions are shown on the System Functions change pages. These pages may be inserted into Volume 5 of the PS 300 Document Set.

The following section contains the NETEDIT Release Notes.

Version A1.V02 - March 1985

NETEDIT V1.08 RELEASE NOTES

A revised version of the NETEDIT programming tool is provided on the magnetic tape distributed with the A2.V02 PS 390 Firmware. A description of changes follows. If you wish a new version of the NETEDIT User's Guide, contact your E&S Account Executive to order the updated documentation.

FORTRAN/Pascal GSR Code Conversion

There are now options to produce FORTRAN or Pascal code, as well as the usual PS 300 ASCII commands, available under CONVERT NETWORK. Selecting these options produces a subroutine or procedure which can be compiled and linked with a user-supplied main program and the appropriate GSR library.

The Pascal code is compatible with VAX/VMS Pascal V2; the FORTRAN code is compatible with VAX/VMS FORTRAN-77.

The menu items ASCII OUTPUT, FORTRAN GSR, and Pascal GSR cause the corresponding type of code to be generated. The other menu items toggle various options on and off; you should set these before you select the item to produce the code.

You must take special care to see that the code for all macros referenced in your network have been converted to the same form (i.e., ASCII, FORTRAN, or Pascal) as the code to be produced for the rest of the network. For example, when you are generating Pascal code you cannot reference an ASCII macro. NETEDIT will give a warning message if you attempt to do this.

The following discussion of how to compile and link the generated code with your program assumes familiarity with the GSRs.

The generated code is output to a file with the same name as the network, with an extension of .PAS for Pascal, and .FOR for FORTRAN. The code is in the form of a single subroutine or procedure with the same name as the network; this routine takes no arguments.

Your program must perform the calls to attach and detach the PS 300 (PAttach/PDetach for Pascal, PATTCH/PDTACH for FORTRAN). You must also supply an error handling routine, as described in the GSR documentation.

For FORTRAN, the error handler must be named ERR. The output file produced by NETEDIT may be compiled independently, or included in a file containing other FORTRAN subprograms. You must then link it with your main program, the error handler, and the FORTRAN GSR library.

For Pascal, the error handler must be named PI_Error_Handler. The suggested method for compilation is to include the file containing the generated code in your main program file, using the %include directive. Your program must also include the declarations in PROCONST.PAS, PROTYPES.PAS, and PROEXTRN.PAS. After compiling the program, you must link it with the Pascal GSR library.

Literal PS 300 Commands Can Be Included in Network

Specially flagged labels can be used to insert random PS 300 commands in a network. Floating comments which start with \+\ or \-\ indicate commands to be inserted before or after the other code for the frame, respectively. These commands are always written to the output file during code conversion, regardless of the SUPPRESS COMMENTS setting.

The statements can be ordered by including a priority number in the flag. For example, statements prefixed with \-1\ are guaranteed to be sent before statements prefixed with \-2\. This is useful for sending an ordered sequence of constants to the same input of a function, for example.

Typically, commands that should be inserted before the other code for a frame are initialize commands or display structure definitions. Commands that should be specified to go at the end of the code for the frame are SETUP CNESS commands, and SEND statements. NETEDIT does not perform any syntax or validity checking on the commands.

Names of functions, variables, and display structures that are referenced in these commands may be prefixed with \F\ and/or \M\ to indicate that the appropriate frame and/or macro prefix should be substituted during code conversion.

Version A1.V02 - March 1985

NETEDIT Now Uses GSRs

NETEDIT has been changed to use the GSR library internally. This should result in some increase in performance for those using high-speed lines. The device type may be specified using the @AttachTo option in the parameter file. The default value, for the RS-232 async line, is:

```
@ATTACHTO logdevnam=tt:/phydevtyp=async
```

See the GSR user's manuals for more information on how to specify this parameter.

If the Pascal GSR library is not available, a library of procedures with the same calls as the GSR routines, but which send the equivalent ASCII commands to the parser, is provided.

Support Network Uses UWFs

Some parts of the support network have been replaced by user-written functions. No new functionality has been added, but users may notice some improvement in performance. NetEdit V1.08 will not work with PS 300 firmware that does not support UWFs (i.e., pre-A1 firmware).

Improved Handling of Arcs

Users should see faster response when adding arcs as a result of changes to the host program and the support network. Adjacent colinear segments are now combined when the arc is processed. In addition, better ways for handling arcs when the items they are attached to have been moved should cut down on the need to manually reroute arcs.

Improved Text Editing Facilities

NETEDIT now uses an improved line editing function for text entry. This function behaves like a one-line screen editor, similar to EMACS in its use of control characters for editing effects. If you are editing an existing piece of text, you do not have to retype the entire line just to make a minor change, as the buffer is initialized to contain the previous contents of the line being edited.

Version A1.V02 - March 1985

The following control characters are used for editing effects:

↑A	Cursor to beginning of line
↑B	Cursor left (back)
↑D	Delete character under cursor
↑E	Cursor to end of line
↑F	Cursor right (forward)
↑K	Delete to end of line
↑R	Retype line
↑U	Delete entire line
DEL	Delete character to left of cursor
RET	Flush buffer

Version A1.V02 - March 1985 (Modified for A2 - April 1987)

Revised Installation Procedures

The PS 300 distribution tape now contains NETEDIT executables as well as source files. This simplifies the installation procedure for sites where no modifications to the source or data files are planned, or where no Pascal compiler is available. Note that the executables were built on a VAX 780 running VMS 3.7, and may not work properly on other versions of the hardware or software.

The procedure for installing NETEDIT without rebuilding it entirely is as follows:

1. Set default to Netedit subdirectory in the A2.V01 subdirectory.

2. Edit NETUSER.COM and change the definition of NETROOT (marked !*INSTALL-DEPENDENT) to the name of the directory created. Make sure this file is readable and executable by all users. See comments in Netuser.Com "Site Customization of Netuser.Com."

3. Copy the empty user log file, NETEDIT0.USR to NETEDIT.USR. Set the protection on this file so that it is writable by all users.

The procedure to install the editor by rebuilding the executables is essentially unchanged. Note that there is an additional !*INSTALL-DEPENDENT parameter in NETBUILD.COM which specifies the directory where the PS 300 Pascal GSR library resides. If you do not have this library, you may use the dummy library supplied on the tape. See comment in NetBuild.Com "Site Customization of NetBuild.Com."

Source files for the user-written functions used by NETEDIT, along with a command file to build the .300 files which may be downloaded to the PS 300, have been provided. However, to rebuild the user-written functions, you must have the Motorola 68000 cross software, which is not supplied by Evans & Sutherland.

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
2A	Hands-on Experience	2	The command to set the line-drawing speed for CSM displays should read: SEND TRUE TO <1> CSM;

2A	Command Language	11	The command to translate the tire primitive to the front left location should read:
			Tran_FL_Tire := TRANSLATE BY .5415,-.1598,
			.3357 APPLIED TO Rot_FL_Tire;

		14	The TRANSLATE command in the definition of the front left snow tire (FL-tire) should read: TRANSLATE BY .5415, -1598, .3357; The remainder of the structure is correct.
		20&23	Figures 4 and 5 are reversed.
		23-28	In BEGIN_STRUCTURE ... END_STRUCTURE commands, there should be no semicolon following "BEGIN_STRUCTURE".
2A	Function Networks 1	4	A new page is supplied to call out the names of interactive nodes in the figure.
		32	The integer 2 should not go to <1>Roty, but to <1>Timer. The command should read: SEND FIX(2) TO <1>Timer;

2A	Viewing Operations	43	In the definition of Top, the viewport command should create a viewport with dimensions 0 to 1 horizontally and vertically. It now creates a viewport from 0 to -1 horizontally and vertically. The viewport command in Top should read:
			VIEWPORT
			HORIZONTAL=0:1
			VERTICAL=0:1;

		45	The names in the last two commands on this page are wrong, they should be Nonsquare_Window and NonSquare. The last two commands should read:
			DISPLAY Nonsquare_Window;
			REMOVE NonSquare;

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
2B	Conditional Referencing	13	IF LEVEL_OF_DETAIL nodes accept integers from 0 to 32767, not from 0 to 14, as stated near the bottom of this page.
		17	Select LEVEL OF DETAIL from the Tutorial Demonstration Programs, not ANIMATED CYLINDER.
2B	Function Networks II	3	A new page is supplied to call out the names of interactive nodes in the figure.
2B	Function Networks II	22	Last command should read: CONNECT Switch7<3> : <1>Rot_Lt_Elbow;
		39&40	Every command must be followed by a semicolon.

2B	Text Modeling	20	Near the top, the command to display the scaled limerick now reads:
			DISPLAY Limerick;
			It should read:
			DISPLAY Scale_Block;

2B	Rendering Operations	17	A replacement page is supplied to correct coordinate values.
		20	In "Object" on the bottom of page 20, polygon 2 is an {outer}, and polygon {5} should be defined as follows: {non-coplanar} POLYGON 1,1,0 1,1,1 1,-1,1 1,-1,0;
3A	Command Summary	Bspline	The node diagram shows inputs <1>, <2>, and <3>. It should show input <1> and input <i>. A real number sent to input <i> changes the knots in the curve. A 2D, 3D, or 4D vector sent to input <i> changes the vertices in the curve.
		Rational Bspline	The node diagram shows inputs <1>, <2>, and <3>. It should show input <1> and input <i>. A real number sent to input <i> changes the knots in the curve. A 2D, 3D, or 4D vector sent to input <i> changes the vertices in the curve.
		Labels	A 3D vector can be sent to input <i> to change the starting location of the i-th label. This is not currently shown.

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
3A	Command Summary	Polynomial	The node diagram shows inputs <1> and <2>. It should show inputs <1> and <i>. An integer sent to input <1> changes the number of chords in the curve and a vector sent to input <i> changes the curve coefficients. The present documentation has this information reversed.
		Rational Polynomial	The node diagram shows inputs <1> and <2>. It should show inputs <1> and <i>. An integer sent to input <1> changes the number of chords in the curve and a vector sent to input <i> changes the curve coefficients. The present documentation has this information reversed.

		XFORM	The 4x4 matrix shown in the example does not contain enough elements and is improperly formatted. In addition, the semicolon is missing from the END_S command. The commands should read:
			`TRAN := BEGIN_S {To be used while getting transformed data}`
			`MATRIX_4x4 1,0,0,0  0,1,0,0  0,0,1,0  0,0,0,1;`
			`INSTANCE OF OBJ;`
			`END_S;`

3A	Function Summary	F:ACCUMULATE	The default values on inputs <3> and <4> are not given. Input <3> defaults to 0 and input <4> defaults to 1.
		F:LINEEDITOR	Output <4> is incorrectly documented as an input to <append> of a Characters node. It should read input <delete> of a Characters node.
		F: MCONCATENATE(n)	This function is misnamed. Its correct name is F:MCAT STRING(n).
		F:STRING_TO_NUM	This function converts a string of digits to a real number. Missing from the current documentation is output <2>. This is a Boolean value which is TRUE if the string received can be converted and false otherwise.
		FKEYS	This initial function instance is listed as an Output function. In fact, it is an Input function.
		TSCSM	This has been erroneously documented as TECSM.

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
3A	Function Summary	F:SEND	Previously undocumented. A new page is supplied.
		F:LIST	Previously undocumented. A new page is supplied.
		ONBUTTONLIGHTS	Previously undocumented. A new page is supplied.
		CSM	Previously undocumented. A new page is supplied.
5	Firmware and Host Software	4	Under "Creating and Downloading the SITE.DAT," Step 3, the correct demuxing character and routing byte to begin the SITE.DAT file is ↑\: (↑ represents the CONTROL value of the \ key). The user-defined commands are entered after these characters and the file must end with the demuxing character and routing byte: ↑\;.
5	Local Data Flow	5-38 (5-62 IMB hosts only)	Table 5-1 Routing Byte Definitions is incomplete and may be misleading. A new page is supplied.

Enhancements in Graphics Firmware Version A2.V01

- This release of the graphics firmware provides the new Writeback feature. The Writeback Feature allows displayed transformed data to be sent back to the host. This feature provides a Writeback command and a Writeback function.

  The Writeback command creates a WRITEBACK operation node and enables the data structure below the node for writeback operations. When the Writeback node is activated, writeback is performed for name1 (the name of the structure for which writeback is applied). A default WRITEBACK operation node is created by the system at initialization time.

  The Writeback Function is initialized by the system and is used to send encoded writeback data to user function networks. This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any writeback operation node in a display structure.

  Writeback is described completely in the Writeback Feature User's Guide, included with this release.

- PVecMax (PVCMax-FORTRAN) has been added to the GSRs. This procedure sets the maximum component of a block-normalized vector list, so that multiple calls may now be made to PVecList for block-normalized vectors.

Modifications in the Graphics Firmware

- Changes to BUTTONSIN (PS 350 Only)

  The initial function instance BUTTONSIN has two new inputs.

  Integer <2> Enable/Disable Bit Mask
  Default FIX(-1) all buttons enabled.

  Boolean <3> TRUE - enable use of bit mask
  FALSE - disable use of bit mask.
  Default FALSE

  The Buttonsin bit mask is a mapping of the bits of a 32-bit integer to the individual buttons. The Most Significant Bit (sign bit) maps to button #1; the least significant bit maps to button #32.

Most Significant Bit ↓ Least Significant Bit ↓

Bits of the Integer	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Button Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

If the bit is set (=1), the button is enable. If the bit is off (=0), the button is disabled.

- Changes to ONBUTTONLIGHTS and OFFBUTTONLIGHTS (PS 350 Only)

  The initial function instance ONBUTTONLIGHTS/OFFBUTTONLIGHTS has one new input.

  New input
  <2> Boolean
  TRUE - interpret integer on input <1> as a bit mask.
  FALSE - interpret integer on input <1> as a button number.

  The ONBUTTONLIGHTS/OFFBUTTONLIGHTS bit mask is a mapping of the bits of a 32 bit integer to the individual buttons. The most significant bit (sign bit) maps to button #1; the least significant bit maps to button #32. If the bit is set (=1) the button light is on.

CHANGE PAGES TO THE DOCUMENT SET OTHER THAN THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND GRAPHICS SUPPORT ROUTINES

CONVERTING INPUT DEVICE VALUES TO UPDATE AN INTERACTION NODE

The first step to selecting the appropriate function to convert input values into values that can update an interaction node is to identify the type of values needed by the node. To understand this, look at the the most common graphics transformations--rotation, scaling, and translation.

Rotations and scales are done with 3x3 matrices; translations are specified with a 2- or 3-dimensional vector. It makes sense, then, that the type of data used by a rotation or scale node is a 3x3 matrix, and the data type for a translation node is a vector.

Your task, if you are trying to rotate part of a model, is to find a way to make an input device, such as a dial, send the correct 3x3 matrices to a rotate node. In this module, this process will be represented by a "black box" (Figure 2) that takes one kind of value and changes it into another kind.

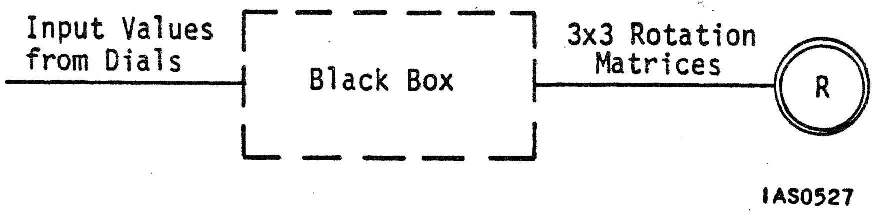


Figure 2. The "Black Box"

In the "Hands-On Experience" module, you created Diamond by specifying a 45 degree rotation of Square. You did not need to work out what the 3x3 matrix for 45 degrees was. Whenever you use a command to create a rotate or scale node (such as Diamond), you only have to specify an angle using a real number value and the PS 300 automatically creates the associated 3x3 matrix.

Once the node is created, however, you can only update it with the type of data it accepts--in this case, a 3x3 matrix. For example, look at the robot display tree again (Figure 3) the names for the interactive nodes are supplied so you can refer to them.

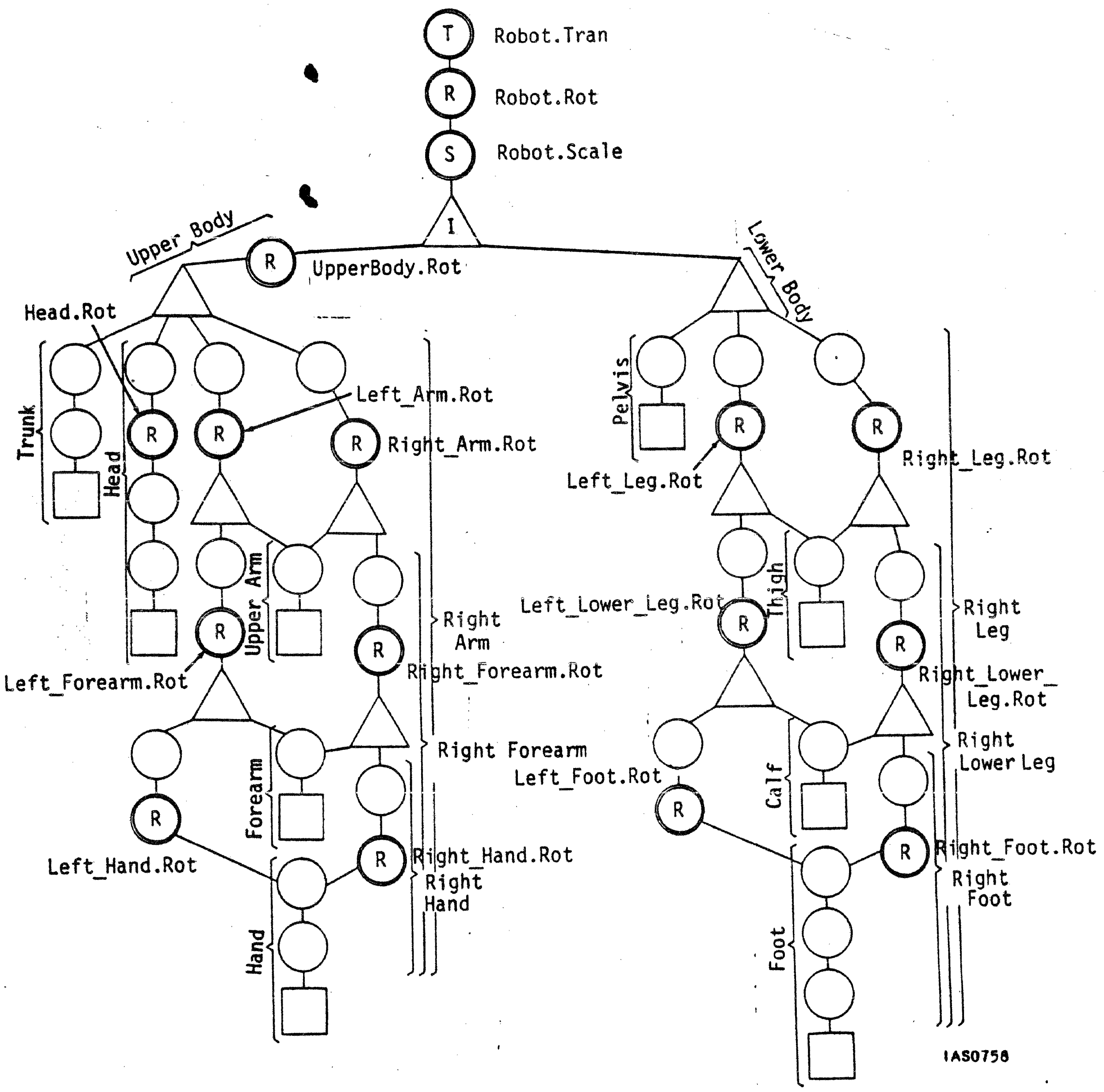


Figure 3. Interactive Nodes in Robot Display Tree

MAKING A SINGLE INPUT DEVICE CONTROL MULTIPLE INTERACTIONS

In "Function Networks I," you constructed a function network for the display tree shown in Figure 1.

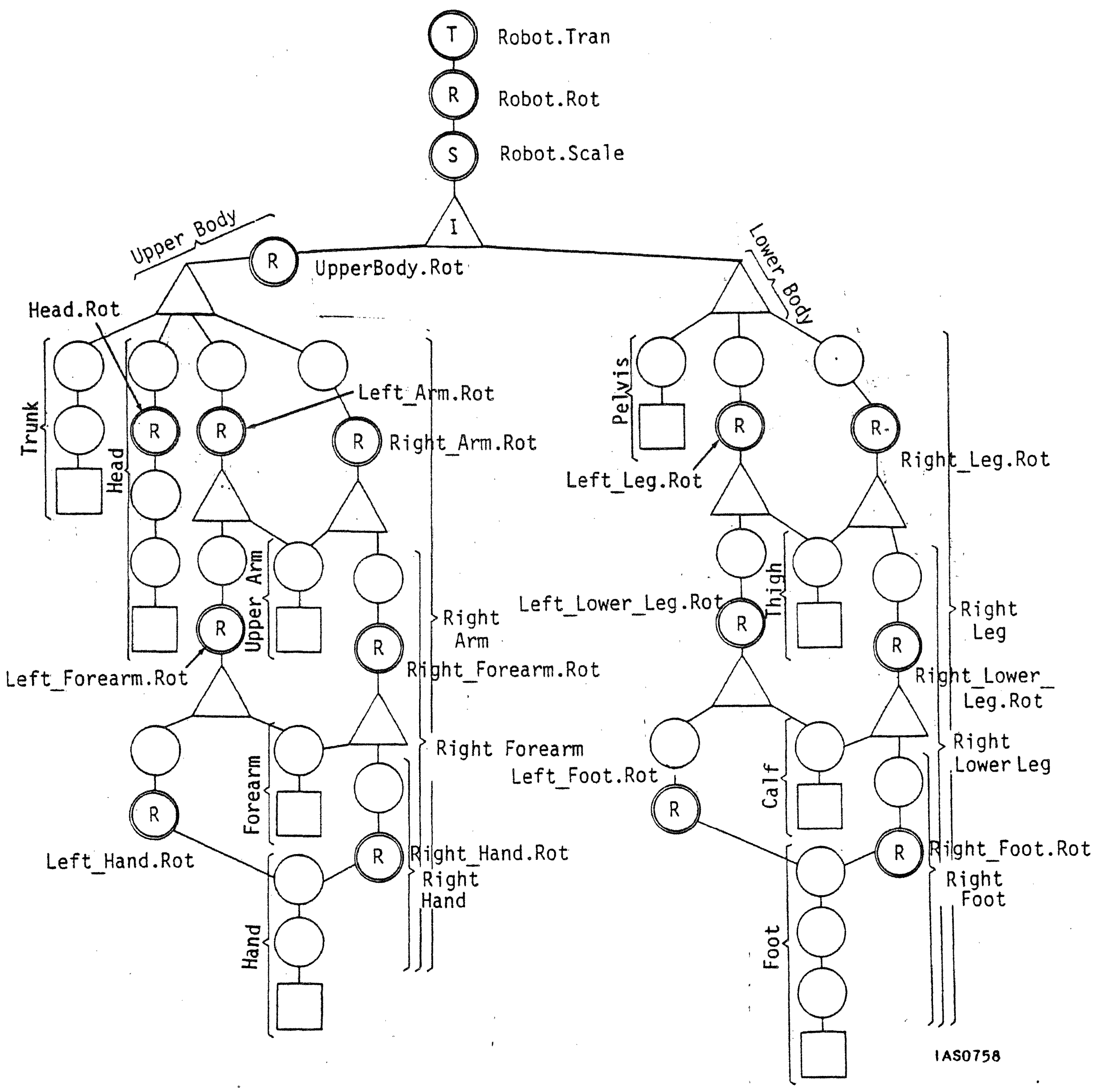


Figure 1. Robot Display Tree

This function network supplied interactions for the top three nodes of the display tree: Robot.Scale, Robot.Rot, and Robot.Tran. Seven dials were required to manipulate the robot: three to rotate it in the X, Y, and Z planes, three to translate it in X, Y, and Z, and one dial to scale the model.

Only one free dial remains, but no other interactive nodes in the robot display tree have yet been connected to functions. To supply X, Y, and/or Z rotations for all the other interactive nodes would require dozens of other dials. This section illustrates how to solve this problem by making one set of eight dials perform like many sets.

The first step in doing this is to determine exactly how many additional dials you will need by deciding how many more interactions in the model you want to control. In addition to Robot.Rot, the robot has 14 rotation nodes. Ten of them require three dials each (three rotations for X, Y, and Z). The two nodes for elbows and the two for knees only use X rotations, requiring only one dial each. The result is a total of 34 additional interactions. To handle these interactions, each dial would have to be connected to about six nodes.

There is nothing to prevent you from connecting a dial to more than one destination. For example, you could hook dial 1, already updating X rotations for the Robot.Rot node, to other rotate nodes. But of course turning that one dial would cause multiple unrelated updates.

Following is one way the dials might logically be assigned to control the interactions.

In Mode 1, the dials would work as presently assigned:

Whole model:	1. Xrot	2. Yrot	3. Zrot	4. Scale
	5. Xtran	6. Ytran	7. Ztran	8. Not Assigned

Mode 2:

Head:	1. Xrot	2. Yrot	3. Zrot	4. Not Assigned
Trunk:	5. Xrot	6. Yrot	7. Zrot	8. Not Assigned

Mode 3:

Right arm:	1. Xrot	2. Yrot	3. Zrot	4. Elbow Xrot
Left arm:	5. Xrot	6. Yrot	7. Zrot	8. Elbow Xrot

Given the following object (Cube):

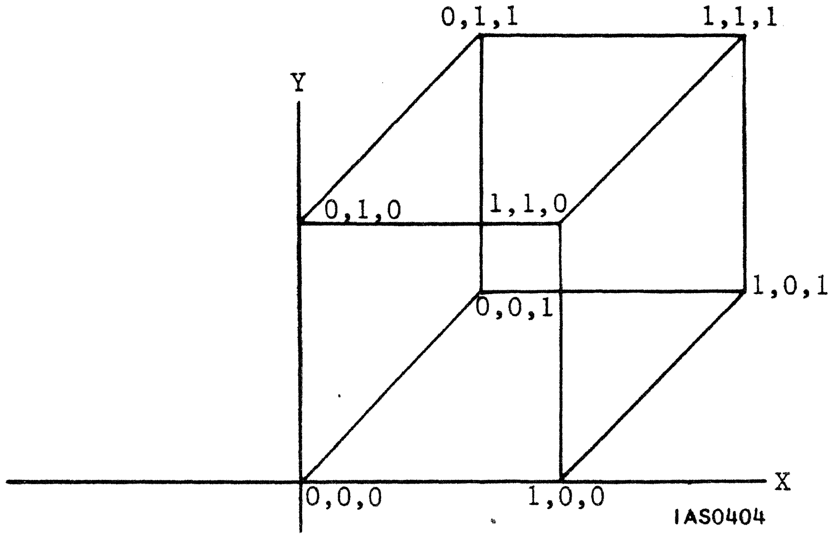


Figure 12. Cube

A correct syntax to define this object is as follows:

```
Cube :=    POLYGON 0,0,0  0,1,0  1,1,0  1,0,0
           POLYGON 1,0,0  1,1,0  1,1,1  1,0,1
           POLYGON 1,1,1  0,1,1  0,0,1  1,0,1
           POLYGON 0,1,1  0,1,0  0,0,0  0,0,1
           POLYGON 0,1,1  1,1,1  1,1,0  0,1,0
           POLYGON 1,0,0  1,0,1  0,0,1  0,0,0;
```

Associating Outer and Inner Contours With COPLANAR

A polygon that represents a face of an object is called an **outer contour.** Some polygons, known as **inner contours** represent cavities, holes, or protrusion sites in an object.

For the PS 340 to interpret inner contours properly, two things must be done. One is to observe the vertex-ordering convention for inner and outer contours. The other is to use the COPLANAR option in the POLYGON clause to associate inner and outer contours.

The vertex ordering rule for inner and outer contours is as follows: vertices of inner contours must run in the opposite sense to the corresponding outer contour. For a solid this implies that the vertices of an inner contour run counterclockwise while outer contours run clockwise when viewed.

The vertices of the following triangular polygon face (outer contour) with a hole in it (inner contour) are ordered as follows.

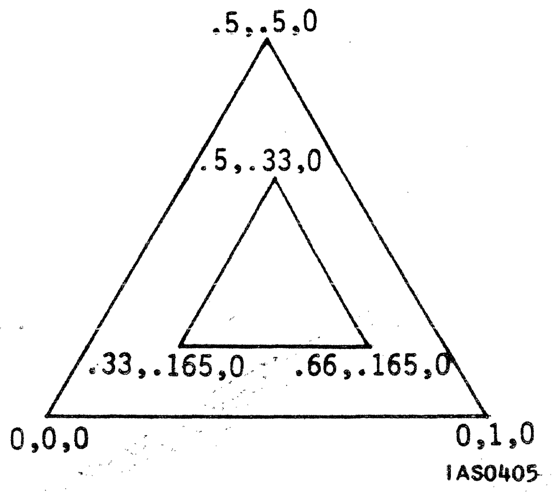


Figure 13. Surface With Inner/Outer Contours

A POLYGON command syntax for this object is :

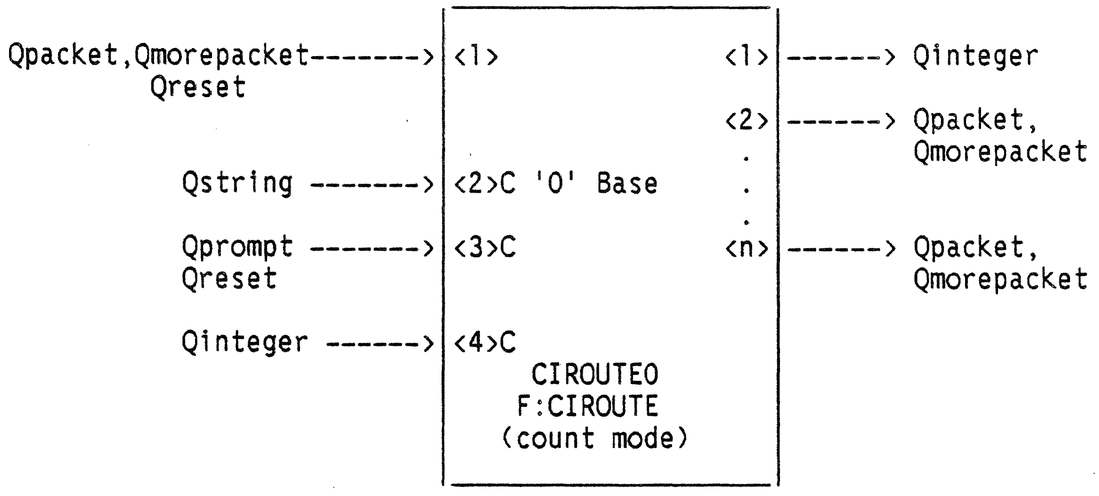
```
Object :=  POLYGON  0,0,0  .5,.5,0  1,0,0 {outer contour}
           POLYGON COPLANAR .5,.33,0  .33,.165,0  .66,.165,0;
           {inner contour}
```

Note that the vertices for the inner contour in the above example are listed in the opposite order of those of the outer contour.

5.2 DATA RECEPTION AND ROUTING NETWORK

F:CIROUTE

Once data have passed through either instance of F:DEPACKET (described in the previous chapter), the next function to receive it is F:CIROUTE. F:CIROUTE has two instances, one for count mode and one for escape mode. They are functionally very similar, and only the count mode instance, CIROUTE0 will be described. CIROUTE0 examines the first character it receives (the character following the count bytes in count mode and the character following the <FS> character in escape mode) to determine where the packet message is to be sent. These characters are "routing" bytes, and are used to select the appropriate channel for data in the PS 300. Data channels include lines to the terminal emulator, the PS 300 command interpreter, the Disk writing function, the Raster function (for PS 340 systems), and other system functions. A base character (defined on Input <2> of CIROUTE0) is subtracted from this routing character before it is used to select the output channel. The base character defaults to the character zero ("0").



The definitions for the inputs and outputs for F:CIROUTE are described in Chapter 6 of this guide.

5.2.1 Routing Byte Definitions

The following table defines the routing bytes and channel parameters for assessing internal PS 300 communication channels.

Table 5-1. Routing Byte Definitions

CIROUTE Output	Routing Byte	Channel Parameter	Description
3	0	1	Parser/Command Interpreter
4	1	2	Command Interpreter via READSTREAM
5	2	3	6-bit binary
6	3	4	Reset network for GSRs
7	4	5	Reserved
8	5	6	Reserved
9	6	7	Download channel for user-written functions
	•		
	•		
	•		
13	:	11	Write ASCII data to diskette
14	;	12	Close file
15	<	13	Write binary data to diskette
16	=	14	Reserved
17	>	15	Channel to terminal emulator
18	?	16	Host message control
19	@	17	Who (used by PSETUP)
20	A	18	Reserved
21	B	19	Raster

NOTE

('?') is the HOST_MESSAGE request channel. <SOP>? followed by ASCII (1 or 2) requests a single message or multiple messages from HOST MESSAGEB.

('@') any message sent on this route triggers the WHO function. (Refer to the PS 300 Host-Resident I/O Subroutine Manual for information on the WHO function.)

5.1.1 Data Flow Overview

The PS 300 accepts data from the IBM host line through the General Purpose Interface Option (GPIO) card. These data will be in two forms; either data for the terminal emulator, or graphical data that have been sent from the host using the cross-compatibility software. The GPIO differentiates between data designated for the terminal emulator and graphics data packaged in the Write Structured Field (WSF) envelopes. The GPIO puts TE data directly into a Screen Buffer in Mass Memory. Graphics data are intercepted by the GPIO for unpacking and repackaging into Qpackets. Routing information is always included at the head of any WSF command. The routing information and the first 238 bytes of data are put into a Qpacket by the GPIO. All subsequent data within the same WSF command are placed into Qmorepackets. When a WSF command is filled to capacity, or a routing change is required, the present WSF command is terminated and a new WSF command is started by a PS 300 low-level communication routine.

The packets of graphical data are passed to the data reception function, F:F_I1_IBM (F:F_I2_IBM). IBMI1$ (an instance of F:F_I1_IBM) allocates new mass memory packet buffers and puts them on a linked list for subsequent use by the GPIO. IBMI1$ passes data through to F:CIROUTE.

F:CIROUTE

CIROUTE0 examines the first character it receives (the character following the count bytes in count mode) to determine where the packet message is to be sent. These characters are "routing" bytes, and are used to select the appropriate channel for data in the PS 300. (Data channels can be chosen by the use of the parameter in the PMuxG GSR Utility Routine. Standard GSR and PSIO calls include embedded routing bytes.) Data channels include lines to the terminal emulator, the PS 300 command interpreter, the disk writing function, the raster function (for PS 340 systems), and other system functions. A base character (defined on Input <2> of CIROUTE0) is subtracted from this routing character before it is used to select the output channel. The base character defaults to the character zero ("0"). The definitions for the inputs and outputs for F:CIROUTE are described in Chapter 6 of this guide.

```
                              _____________________
Qpacket,Qmorepacket-------> |<1>             <1>| ------> Qinteger
          Qreset            |                  .|
                            |                <2>| ------> Qpacket,
                            |                  .|         Qmorepacket
        Qstring --------> |<2>C '0' Base     .|
                            |                  .|
        Qprompt --------> |<3>C              <n>| ------> Qpacket,
        Qreset              |                   |         Qmorepacket
        Qinteger -------> |<4>C                |
                            |      CIROUTE0    . |
                            |      F:CIROUTE     |
                            |____________________|
```

5.1.2 Routing Byte Definitions

The following table defines the routing bytes and channel parameters for assessing internal PS 300 communication channels.

Table 5-1. Routing Byte Definitions

CIROUTE Output	Routing Byte	Channel Parameter	Description
3	0	1	Parser/Command Interpreter
4	1	2	Command Interpreter via READSTREAM
5	2	3	6-bit binary
6	3	4	Reset network for GSRs
7	4	5	Reserved
8	5	6	Reserved
9	6	7	Download channel for user-written functions
	•		
	•		
	•		
13	:	11	Write ASCII data to diskette
14	;	12	Close file
15	<	13	Write binary data to diskette
16	=	14	Reserved
17	>	15	Channel to terminal emulator
18	?	16	Host message control
19	@	17	Who (used by PSETUP)
20	A	18	Reserved
21	B	19	Raster

NOTE

('?') is the HOST_MESSAGE request channel. '?' followed by ASCII (1 or 2) requests a single message or multiple messages from HOST MESSAGEB.

('@') any message sent on this route triggers the WHO function.

F:IBM_KEYBOARD

```
                 ________________________
                |                        |
                |     F:IBM_KEYBOARD     |
                |                        |
Qpacket----->   |<1>                  <1>|----->Qpacket
                |                        |
QBoolean --->   |<2>                  <2>|----->Qinteger
                |                        |
                |                     <4>|----->Qpacket
                |                        |
                |                     <3>|----->Qpacket
                |                        |
                |                     <5>|----->Qpacket
                |                        |
                |                     <6>|----->QBoolean
                |                        |
                |                     <7>|----->Qpacket
                |                        |
                |                     <8>|----->QBoolean
                |                        |
                |                    <10>|----->QBoolean
                |     (IBMKBD1)          |
                |                    <11>|----->QBoolean
                |________________________|
```

F:IBM_KEYBOARD accepts character packets from the keyboard on input <1> and, based on the mode selected by the mode keys (either the LINE LOCAL key or the HOST, LOCAL and COMMAND keys, depending on the type of keyboard used), outputs packets for use by the function network, the line editor, or an IBM host. Packets of characters for the function KEYBOARD are output on output <1>. Qintegers to be sent to the function FKEYS are output on output <2>. Qpackets of characters to be sent to the function SPECKEYS are output on output <3>. Qpackets of characters for the line editor are output on output <4>. Qpackets of IBM scan codes for an IBM host are output on output <5>. A QBOOLEAN TRUE used to trigger the hardcopy functions is output on either output <6>, output <10>, or output <11>, based on the mode of the keyboard.

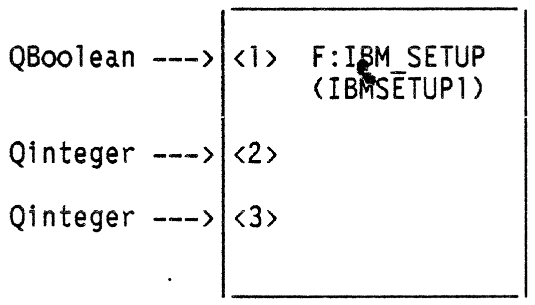
A TRUE used to trigger the loading of the IBM 3250 function network is output on output <7> when IBM 3250 mode is selected while in SETUP mode.

A TRUE used to trigger the deletion of the IBM 3250 function network is output on output <8> when the PS 300 mode is selected while in SETUP.

Input <2> accepts a Boolean that indicates which type of keyboard is being used.

True = IBM style keyboard
False = VT100 style keyboard

F:IBM_SETUP



F:IBM_SETUP is used to change the parameters used by the IBM communications. Input <1> accepts an integer that specifies the maximum number of packets that can be in the pool of empty input packets.

F:IBM_SETUP is used to change certain values used by the IBM communications.

Input <1> is used to trigger the function.

Input <2> is used to specify the number of empty I/O input packets that are to be maintained in the I/O input pool.

Input <3> is used to specify the device address when an IBM 3250 interface is being used.

F:CI

```
                  |        F:CI           |
Qchopitems ---->  | <1>              <1>  | ----> unused
Qprompt           |                       |
                  |                  <2>  | ----> unused
                  |                       |
                  |                  <3>  | ----> error messages
                  |                       |
                  |                  <4>  | ----> Qboolean
                  |                       |
                  |                  <5>  | ----> Qprompt
                  |                       |
                  |                  <6>  | ----> unused
                  |       (H_CIO)         |
                  |       (CIO)           |
```

This function interprets commands, creating display structures and function networks. It receives input either from a chop/parse function or a Readstream function (if using the GSRs).

A single parameter is given when this function is instanced (for example H CI0:=F:CI(4);). This parameter is the "CINUM" and is used to identify all names and connections this CI makes. When the CI receives an INIT command, it destroys only those connections it has made and only those structures associated with the names which have its CINUM.

Note: A name is created when that name is referenced for the first time, even if it has no associated structure. The CI that created the name is the "owner" of that name, even if the entity it refers to is created by another CI.

Note: Each function has an output <0> that is used to send error messages (such as illegal input error messages). The connection from this output is made automatically by the CI that creates the function. The CI finds the appropriate error function to connect output <0> to by looking on its own output <3>.

Output 4 sends out a Qboolean with a TRUE value when an INIT command is entered. This output is connected to the initial function CLEAR_LABELS to clear out the labels on the keyboard and dials.

7.5.3 Command Status Command

The command:

COMMAND STATUS;

directs the command interpreter to print the status of the command stream. The message output lists the number of open BEGIN...END and BEGIN_STRUCTURE...END_STRUCTURE commands, and indicates if the privileged state is operative. The message also indicates if the optimize structure model is in effect.

7.5.4 Reboot Command

The command

REBOOT password;

reboots the PS 300 as if from power-up. If no password has been setup, then any character string will do. Otherwise entering an incorrect password will give an error message. The REBOOT command can appear anywhere; it can occur within BEGIN...END and BEGIN_STRUCTURE...END_STRUCTURE as well as without. It may be named or not. However, it cannot be within a quote or comment.

The command causes the PS 300 to reboot just as if it had been powered up (starts the confidence tests at "A", etc.).

7.5.5 Set Priority

The command

Set Priority of name to i;

sets the execution priority of a function (name) to some integer (i) between 0 and 15. All user instancible functions and most functions instanced by the system at boot time have a default value of 8. Lowering a function's priority number raises its priority and causes it to run before any functions with a larger number. A typical use of this command is to give to a function a priority number greater than 8 so it runs only when no other functions are running (i.e. functions at default priority 8). Assigning priority numbers less than 8 could be potentially very "dangerous," since their execution could lock up the system.

Since this command will affect the execution of other functions in a function network, careful consideration must be given to its use. E&S does not recommend the use of this procedure by anyone who does not have a complete understanding of functions and their interrelationships.

7.5.6 Notes On Using the CONFIGURE Mode

E&S reserves the right to change the content of the CONFIG.DAT file and the implementation of the CONFIG.DAT file without prior notice. Use of any named entities or networks instanced in CONFIGURE mode that have names identical to any names found in the CONFIG.DAT file will result in unpredictable system behavior. E&S will not use any names that are preceded with the three characters CM_.

TABLE 9-1 PS 300 TRAPS and Their Meanings

NUMBER	DEFINITION
0	Not enough available memory to come up or handle request.
1	E&S firmware error.
2	Memory corrupted or over-written (could be caused by UWF).
5	Attempt to wait on queue when function is waiting on another device (CLOCK, I/O)(could be caused by UWF).
6	System errors (see Table 3).
8	Mass memory error if address on LEDs is between 200 and 300; unexpected interrupt on a vector with no routine, if address is between 300 and 400. For example, if address on LEDs is 22C, error occurred on memory card 200000-300000. If address is 23C, error occurred on memory card 300000-400000 and so forth.
9	Utility routine was called which was not included in system link.
10	Memory corrupted or over-written (could be caused by UWF).
11	E&S firmware error.
12	Pascal in-line runtime error: usually caused by Case statement in Pascal with no Otherwise clause (could be caused by UWF).

Version A2.V02

F:READDISK

```
                     _______________________________
                    |                               |
                    |          F:READDISK           |
                    |                               |
    Qpacket    ---->|<1>                         <1>|---->  Qpacket
                    |                               |
   QBoolean    ---->|<2>                         <2>|---->  QBoolean
                    |                               |
                    |(Readascii1 for ASCII file)    |
                    |(Readbinary1 for binary)       |
                    |_______________________________|
```

This function reads a file from the floppy disk and sends the data out output <1> in Qpackets. Input <1> accepts a Qpacket of 1 to 8 characters specifying the name of the file to be read. All disk drives are searched for the file until found; if the file is not found, an error message is produced.

A True on input <2> tells the function to delete the file after reading. Input <2> is a constant input queue and is initialized to False.

A True is output from <2> when the file is found and read successfully. A False is output when the file is not found.

Note: The file name sent on input <1> should not include the file extension. The file on the disk must have the extension ".DAT".

CHANGE PAGES TO THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND GRAPHICS SUPPORT ROUTINES

(Change pages exclusive to the Rendering Option are supplied with the PS 390 Rendering Release Notes.)

FORMAT

```
name := LOAD VIEWport HORizontal = hmin:hmax
             VERTical = vmin:vmax
             [INTENsity = imin:imax] [APPLied to name1];
```

DESCRIPTION

The LOAD VIEWPORT command for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standard PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It affects all objects below the node created by the command in the display tree.

PARAMETERS

hmin,hmax,vmin,vmax – The x and y boundaries of the new viewport. Values must be within the –1 to 1 range.

imin,imax – Specifies the minimum and maximum intensities for the viewport. imin is the intensity of lines at the back clipping plane; imax at the front clipping plane. Values must be within the 0 to 1.

name1 – The name of the structure to which the viewport is applied.

DEFAULT

The initial viewport is the full PS 300 screen with full intensity range (0 to 1) using the standard PS 300 Viewport command.

```
VIEWport HORizontal = -1,1 VERTical = -1,1 INTENsity = 0:1;
```

(continued)

NOTES

A new VIEWport is not defined relative to the current viewport, but to the full PS 300 screen.

If the viewport aspect ratio (vertical/horizontal) is different from the window aspect ratio (y/x) or field-of-view aspect ratio (always 1) being displayed in that viewport, the data displayed there will appear distorted.

DISPLAY TREE NODE CREATED

This command creates a load viewport operation node that has the same inputs as the standard viewport operation node. The matrix contained in this node is not concatenated with the previous viewport matrix.

NOTES ON INPUTS

1. For 2x2 matrix input, row 1 contains the hmin,hmax values and row 2 the vmin,vmax values.

2. For 3x3 matrix input, column 3 is ignored (there is no 3x2 matrix data type), rows 1 and 2 are as for the 2x2 matrix above, and row 3 contains the imin,imax values.

FORMAT

name := RAWBLOCK i;

DESCRIPTION

Used to allocate memory that can be directly managed by a user-written function or by the physical I/O capabilities of the Parallel or Ethernet Interfaces.

PARAMETERS

i – bytes available for use.

NOTES

1. The command carves a contiguous block of memory such that there are "i" bytes available for use.

2. The block looks like an operation node to the ACP. The descendant alpha points to the next long word in the block. What the ACP expects in this word is the .datum pointer of the alpha block. (The datum pointer points to the first structure to be traversed by the ACP. This is the address in memory where the data associated with a named entity is located.)

3. To use this block, the interface or user-written function fills in the appropriate structure following the .datum pointer. When this is complete, it changes the .datum pointer to the proper value and points to the beginning of the data. After the ACP examines this structure, it displays the newly-defined data. (Use the ACPPROOF procedure to change the .datum pointer with a user-written function.)

4. More than one data structure at a time can exist in a RAWBLOCK. It is up to the user to manage all data and pointers in RAWBLOCK.

5. A RAWBLOCK may be displayed or deleted like any other named data structure in the PS 300. When a RAWBLOCK is returned to the free storage pool, the PS 300 firmware recognizes that it is a RAWBLOCK and does not delete any of the data structures linked to RAWBLOCK.

DISPLAY TREE NODE CREATED

Rawblock data node.

FORMAT

```
name := SET BLINKing switch
        [APPLied to name1];
```

DESCRIPTION

This command turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

switch - Boolean value. TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

name1 - The name of the structure that will be affected by the command.

NOTE

PS 330 style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

DISPLAY TREE NODE CREATED

This command creates a set blinking on/off operation node in the display structure that determines whether blinking will occur in the objects positioned below it in the display structure.

INPUTS FOR UPDATING NODE

The blinking on/off operation node can be modified by sending a Boolean value to input <1>.

FORMAT

```
name := SET BLINK RATE n
            [APPLied to name1];
```

DESCRIPTION

This command specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

n – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for n refreshes and off for n refreshes.

name1 – The name of the structure to which the blinking rate is applied.

NOTE

PS 330 style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

DISPLAY TREE NODE CREATED

This command creates a set blinking rate operation node in the display tree that specifies the blinking rate for all objects below it.

INPUTS FOR UPDATING NODE

The node can be modified by sending an integer to input <1> which will change the blinking rate.

FORMAT

```
name := SET LINe_texture [AROUnd_corners] pattern
                            [APPLied to name1];
```

DESCRIPTION

Specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter **pattern** is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7 bit positions in **pattern** when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	--------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners - Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at vector endpoints.

pattern - An integer between 1 and 127 that specifies the desired line texture. When **pattern** is less that 1 or greater than 127, solid lines are produced.

name1 - The name of the structure to which the line texture is applied.

(continued)

DEFAULTS

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather than solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The **With Pattern** and **curve** commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

NODE CREATED

This command creates a line texture operation node with line texture to be applied to all vectors below in the display structure hierachy. Sending a Boolean value to input <1> of the node turns the continuous texture feature on or off. Sending an integer value to the node changes the pattern.

FORMAT

name := VECtor_list [options] [N=n] vectors;

DESCRIPTION

Defines an object by specifying the points comprising the geometry of the object and their connectivity (topology).

PARAMETERS

name - Any legal PS 300 name.

options - Can be none, any, or all of the following five groups (but only one from each group, and in the order specified):

1. BLOCK_normalized - All vectors will be normalized to a single common exponent.

2. COLOR - This option is used when specifying color-blended vectors (refer to SET COLOR BLENDing command) to indicate that vector colors will be specified in lieu of vector intensities. When the COLOR option is used, the optional I=i clause used to specify the intensity of a vector (refer to the vectors parameter below) is replaced by the optional H=hue clause, where H is a number from 0 to 720 specifying the individual vector hues. The default is 0 (pure blue).

   The 0-720 scale for the H=hue clause is simply the SET COLOR scale of 0-360 repeated over the interval 360-720. On this scale, 0 represents pure blue, 120 pure red, 240 pure green, 360 pure blue again, 480 pure red again, 600 pure green again, and 720 pure blue. This "double color wheel" allows for color blending either clockwise or counterclockwise around the color wheel.

3. Connectivity:

   A. CONNECTED_lines - The first vector is an undisplayed position and the rest are endpoints of lines from the previous vector.

PARAMETERS (continued)

B. SEParate_lines - The vectors are paired as line endpoints.

C. DOTs - Each vector specifies a dot.

D. ITEMized - Each vector is individually specified as a move to position (P) or a line endpoint (L).

E. TABulated - This clause is used to specify an entry into a table that is used for specifying colors for raster lines and for specifying colors, radii, diffuse, and specular attributes for raster spheres. This option is also used to alter the attribute table itself.

When the TABulated option is used, the T=t clause replaces the I=i clause (for intensities) and the H=hue clause (for vector hues). The default is 127 (table entry 127).

There are 0 to 127 entries into the Attribute table. The Attribute table may be modified via input <14> of the SHADINGENVIRONMENT function.

4. Y and Z coordinate specifications (for constant or linearly changing Y and/or Z values):

   Y = y[DY=delta_y][Z = z[DZ=delta_z]]

   where y and z are default constants or beginning values, and delta_y and delta_z are increment values for subsequent vectors.

5. INTERNAL_units - Vector values are in the internal PS 300 units [LENGTH]. Specifying this option speeds the processing of the vector list, but this also requires P/L information to be specified for each vector, and it doesn't allow default y values or specified intensities.

n - Estimated number of vectors.

PARAMETERS (continued)

vectors -The syntax for individual vectors will vary depending on the options specified in the options area. For all options except ITEMized, COLOR, and TABulated the syntax is:

```
xcomp[,ycomp[,zcomp]][I=i]
```

where xcomp, ycomp and zcomp are real or integer coordinates and i is a real number ($0.0 \leq i \leq 1.0$) specifying the intrinsic intensity for that point (1.0 = full intensity).

For ITEMized vector lists the syntax is:

```
P xcomp[,ycomp[,zcomp]][I=i]
```

or

```
L xcomp[,ycomp[,zcomp]][I=i]
```

where P means a move-to-position and L means a line endpoint.

If default y and z values are specified in the options area, they are not specified in the individual vectors.

For color-blended (COLOR) vector lists, the syntax is:

```
xcomp[,ycomp[,zcomp]][H=hue]
```

where xcomp, ycomp and zcomp are real or integer coordinates and hue is a real number between 0 and 720 specifying the hue of a vector.

For TABulated vector lists (TAB), the syntax is:

```
xcomp[,ycomp[,zcomp]][T=t]
```

where t is an integer between 0 and 127 specifying a table entry.

DEFAULTS

If not specified, the options default to:

1. Vector normalized
2. Not color blended
3. Connected
4. No default y or z values are assumed (see note 5)
5. Expecting internal units

Non color-blended vectors default to:

```
xcomp,ycomp[,zcomp][I=i]
```

If i is not specified, it defaults to 1.

Color-blended vectors default to:

```
xcomp,ycomp[,zcomp][H=hue]
```

If hue is not specified, it defaults to 0 (pure blue).

Tabulated vectors default to:

```
xcomp,ycomp[,zcomp][T=t]
```

If the table entry is not specified, it defaults to 127 (table entry 127).

NOTES

1. If n is less than the actual number of vectors, insufficient allocation of memory will result; if greater, more memory will be allocated than is used. (The former is generally the more severe problem.)

2. All vectors in a list must have the same number of components.

3. If y is specified in the options area, z must be specified in the options area.

NOTES (continued)

4. If no default is specified in the options area and no z components are specified in the vectors area, the vector list is a 2D vector list. If a z default is specified in the same case, the vector list is a 3D vector list.

5. The first vector must be a position (P) vector and will be forced to be a position vector if not.

6. Options must be specified in the order given.

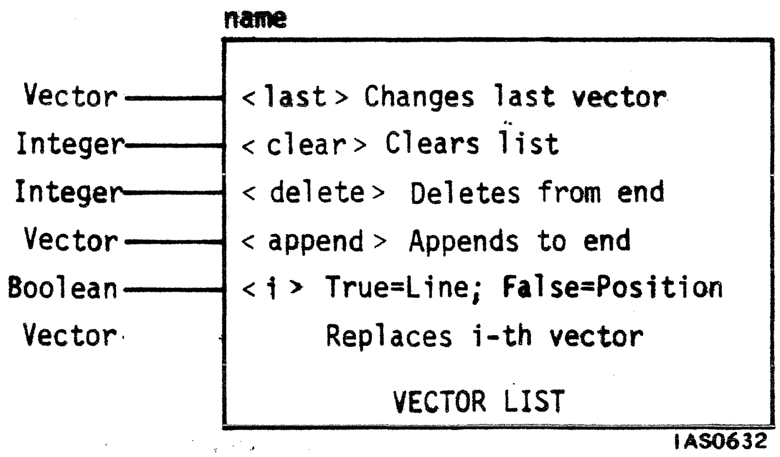
7. If CONNECTED_lines, SEParate_lines, or DOTs are specified in the options area but the vectors are entered using P/Ls, then the option specified takes precedence.

8. Block normalized vector lists generally take longer to process into the PS 300, but are processed faster for display once they are in the system.

DISPLAY TREE NODE CREATED

Vector list data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. Vector list nodes are in one of two forms:

   A. If DOTs was specified in the options area of the command, a DOT mode vector list node is created. The Boolean input to <i> is ignored in this case as well as the P/L portion of input vectors, and all vectors input are considered new positions for dots.

   B. All other vector list nodes created can be considered to be 2D or 3D ITEMized with intensity specifications after each vector, and if a 3D vector is input to a 2D vector list node, the last component modifies the intensity.

2. If a 2D vector is sent to a 3D vector list, the z value defaults to 0.

3. When you replace the i-th vector, the new vector is considered a line (L) vector unless it was first changed to a position vector with F:POSITION_LINE.

EXAMPLES

```
A := VECtor_list BLOCK SEParate INTERNAL N=4
     P 1,1 L -1,1 L -1,-1 L 1,-1;

B := VECtor_list n=5
     1,1 -1,1 I=.5
     -1,-1 1,-1 I=.75
     1,1;

C := VECtor_list ITEM N=5
     P 1,1
     L -1,1
     L -1,-1
     P 1,-1
     L 1,1;

D := VECtor_list TABulated N=5  {for drawing raster lines}
     P 0,1,0
     L 0,0,0  t=5
     L 1,0,0  t=2
     P 1,1,0  t=3
     L 0,1,0  t=4;
```

Version A2.V01

FORMAT

```
name := WRITEBACK [APPLied to name1];
```

DESCRIPTION

The WRITEBACK command creates a WRITEBACK operation node and delineates the data structure below the node for writeback operations. When the WRITEBACK operation node is activated, writeback is performed for name1.

PARAMETERS

name1 - The name of the structure or node to which writeback is applied.

NOTES

1. This node delimits the structure from which writeback data will be retrieved. Only the data nodes that are below the WRITEBACK operation node in the data structure will be transformed, clipped, viewport scaled, and sent back to the host.

2. Only a structure that is being displayed can be enabled for writeback. This means that the WRITEBACK operation node must be traversed by the display processor and so must be included in the displayed portion of the structure. If the writeback of only a portion of the picture is desired, WRITEBACK nodes must be placed appropriately in the display structure.

3. Any number of WRITEBACK nodes can be placed within a structure. Only one writeback operation can occur at a time. If more than one node is triggered, the writeback operations are performed in the order in which the corresponding nodes were triggered. If the user creates any WRITEBACK nodes (other than the WRITEBACK node created initially at boot-up), these nodes must be displayed before being triggered. If the nodes are triggered before being displayed, an error message will result.

4. The terminal emulator and message_display data will not be returned to the host.

DISPLAY TREE NODE CREATED

The command creates a WRITEBACK operation node.

Table 1. Key to Abbreviations for Valid Data Types

KEY TO VALID DATA TYPES	
Any	Any message
B	Boolean value
C	Constant value
CH	Character
I	Integer
Label	Data input to LABELS node
M	2x2, 3x3, 4x3, 4x4 matrix
PL	Pick list
R	Real number
S	Any string
Special	Special data type
V	Any vector
2D	2D vector
3D	3D vector
4D	4D vector
2x2	2x2 matrix
3x3	3x3 matrix
4x3	4x3 matrix
4x4	4x4 matrix

Conjunctive/Disjunctive Sets

Some PS 300 functions have conjunctive or disjunctive inputs and outputs. A function with conjunctive inputs must have a new message on every input before it will activate. A function with conjunctive outputs will send a message on every output when the function is activated.

Conversely, a disjunctive-input function does <u>not</u> require a new message on every input to activate. A disjunctive-output function may not send a message on each output (or any output) every time it receives a complete set of input messages.

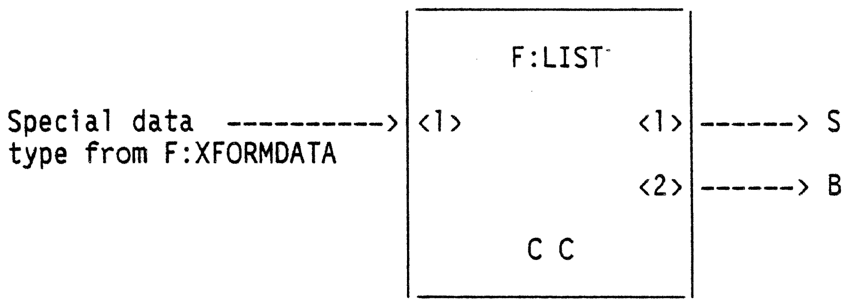
The F:ADD function, for example, has conjunctive inputs. A value must be sent to each of the two inputs before the function will fire. The inputs are then added together, which produces an output that is the sum of the inputs. The output is conjunctive. Unlike F:ADD, F:ADDC is a disjunctive-input function; it does not require a new message on every input.

F:BROUTE, on the other hand, is a conjunctive-input, disjunctive-output function. Both inputs require messages to activate the function. However, a message will be sent out only one of the outputs, depending on the value received on input 1.

F:ACCUMULATE is an example of different sort of disjunctive output. Every input does not produce an output. The function activates each time a new message is received on input 1, but the output fires at specified intervals rather than each time the function is activated.

The following notation is used in the Function Summary to indicate conjunctive and disjunctive inputs and outputs.

KEY TO CONJUNCTIVE/DISJUNCTIVE SYMBOLS	
CC	conjunctive inputs, conjunctive outputs
CD	conjunctive inputs, disjunctive outputs
DC	disjunctive inputs, conjunctive outputs
DD	disjunctive inputs, disjunctive outputs



PURPOSE

Converts the output of the F:XFORMDATA function to an ASCII string. This function is always used with F:XFORMDATA.

DESCRIPTION

INPUT
<1> - data output by F:XFORMDATA

OUTPUT
<1> - resulting ASCII string
<2> - Boolean (TRUE)

DEFAULTS

None.

NOTES

1. Input <1> is always connected to output <1> of F:XFORMDATA.

2. Output <2> is TRUE when processing is complete. There is no output otherwise.

3. Output <2> should be connected to an instance of F:SYNC(2) to synchronize F:LIST completion with the initiation of a subsequent transformed-data request.

```
                   ________________
                  |
                  | F:CONCATXDATA(N)
                  |
XFORMDATA1---->   | <1>          <1> |-----> to SOLID_RENDERING
                  |                  |
XFORMDATA2---->   | <2>              |
                  |                  |
                  |                  |
XFORMDATA----->   | <N>              |
                  |__________________|
```

PURPOSE

Accepts up to 127 transformed vector lists (output from XFORMDATA functions) and concatenates them into a single transformed vector list.

DESCRIPTION

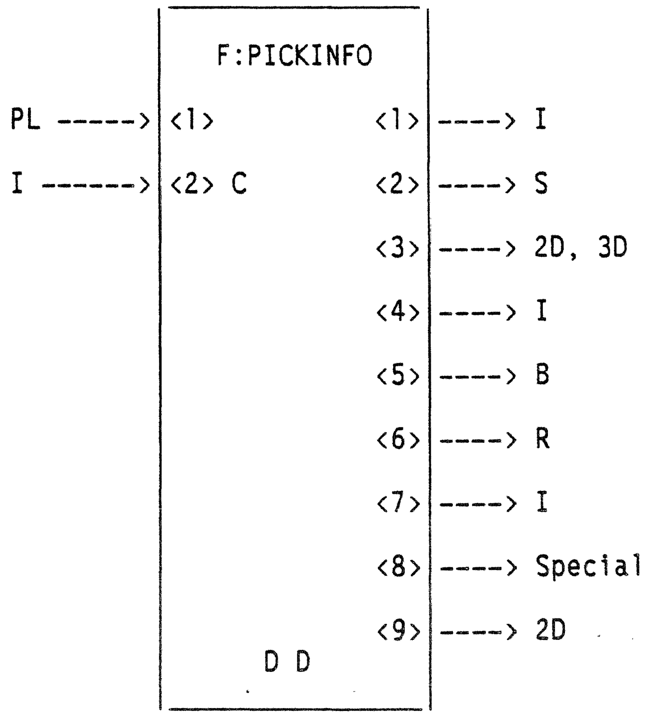
INPUT
<1> - output of F:TRANSFORMDATA (transformed vector list)
.
.
.
<N> - output of F:TRANSFORMDATA (transformed vector list)

OUTPUT
<1> - concatenated vector list

NOTES

1. This function is used to avoid the maximum vector restriction imposed on the output of F:XFORMDATA. The XFORMDATA function will return a maximum of 2048 vectors. To obtain a rendering on the PS 340 raster display of greater than 2048 vectors, the output of multiple instances of XFORMDATA must be concatenated into a single transformed vector list which can be sent to the rendering node.

2. Inputs <1> through <N> accept a transformed vector list output from F:XFORMDATA.



PURPOSE

Reformats picklist information for use by other functions. The output picklist is separated into its component parts.

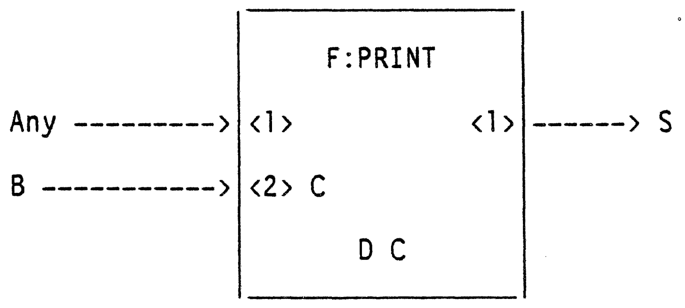
DESCRIPTION

INPUT
<1> - picklist
<2> - depth within structure reported (constant)

OUTPUT
<1> - index
<2> - pick identifier(s)
<3> - coordinates
<4> - dimension
<5> - coordinates reported
<6> - curve parameter, t
<7> - data type code
<8> - name of picked element
<9> - screen coordinates of the picked point

PS 350 Modifications

Output <9> has been added to F:PICKINFO. This output reports the screen coordinates of a pick.



PURPOSE

Converts any data type to string format; that is, it performs an inverse of the operation that occurs when an ASCII string is input to the PS 300 and is converted to one of the data types.

DESCRIPTION

INPUT
<1> - any message
<2> - Boolean governing numeric format (constant)

OUTPUT
<1> - string

PS 350 Modification

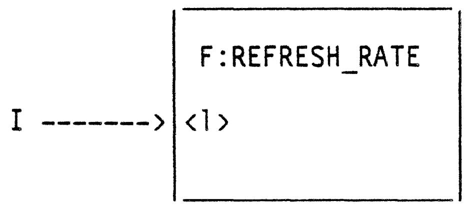
Screen coordinates, if passed to the function from F:PICKINFO, are added to the string output on <1>. Output <1> has been modified to report a pick in which coordinate picking information is given:

For a vector declared in a VECTOR_LIST, the output string format is:

<1><dimension><pick_x><pick_y>[<pick_z>]<t>
<pick ID's><screen_x><screen_y>

For a vector within a polynomial curve the output string format is:

<2><dimension><pick_x><pick_y>[<pick_z>]<t>
<pick ID's><screen_x><screen_y>

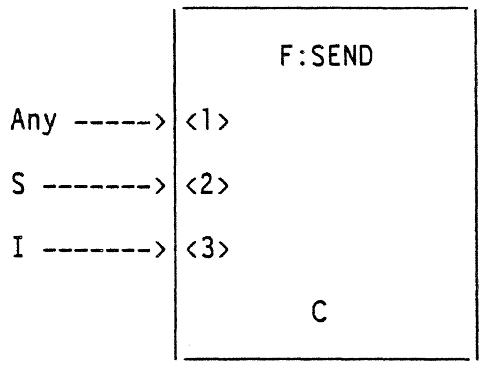


PURPOSE

Locks refresh rate. This function accepts an integer on input <1>. The integer must be in the range of 2 through 5. This is the number of ticks per refresh frame (ticks occur at twice the line frequency). The actual refresh rate depends on the line frequency.

Ticks	60Hz	50Hz
2	60	50
3	40	33
4	30	25
5	24	20

Version A1.V02



PURPOSE

This is the function network equivalent of the SEND command. It allows you to send any valid data type to any named entity at any valid index.

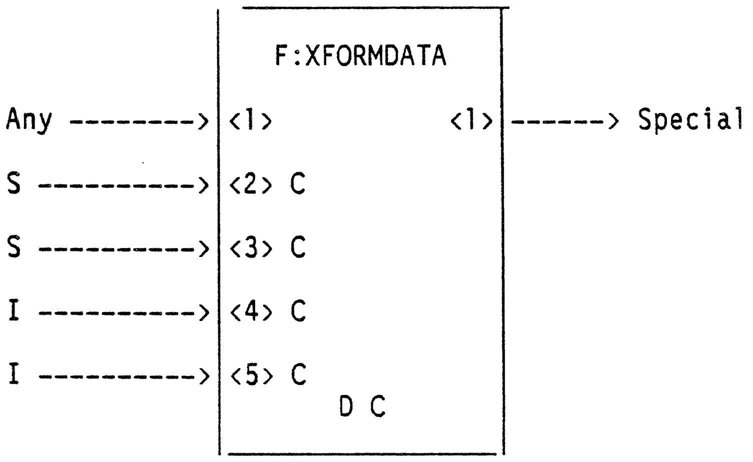
DESCRIPTION

INPUT
<1> – message sent
<2> – name of the destination node
<3> – index into the destination node

NOTES

1. This function has no output.
2. Input <1> accepts special data types that most functions do not accept, such as the data type output by F:LABEL.
3. The SETUP CNESS command can be used to specify constant inputs as default values.

Version A2.V01



PURPOSE

Sends transformed data (either a vector list or a 4x4 matrix) to a specified destination (e.g., the host, a printer, or the screen).

DESCRIPTION

INPUT
<1> – any message
<2> – name of XFORM node (constant)
<3> – name of destination object (constant)
<4> – destination vector index (constant)
<5> – number of vectors (constant)

OUTPUT
<1> – special data type used exclusively as input to F:LIST

DEFAULTS

Default for input <4> is 1, default for input <5> is 2048.

NOTES

1. Input <1> is a trigger for F:XFORMDATA. This input would typically be connected to a function button, either directly or via F:SYNC(2), allowing transformed data to be requested easily.

2. Input <2> is a string or matrix containing the name of the XFORM command in the display tree (either XFORM MATRIX or XFORM VECtor). By referring to an XFORM command, this input indirectly specifies the object whose transformed data is to be sent. If the string names something other than an XFORM command, an error message is displayed. If the string names a node which does not exist, an error message is sent and the message is removed from input <2>.

3. Input <3> is a string containing the name to be associated with the transformed vectors. The name need not be previously defined. If this input does not contain a valid string, the transformed matrix or vectors will be created without a name (an acceptable situation unless the transformed vectors need to be referenced or displayed.) The transformed vector list can be displayed or modified, provided a name is given on this input. The transformation matrix cannot be used, however, so naming and sending it to input <3> is not useful.

4. Input <4> is an integer index specifying the place in a vector list at which the PS 300 is to start returning transformed data. This input is only used when the command name at input <2> represents an XFORM VECtor command (not an XFORM MATRIX command). The default value is 1.

5. Input <5> is an integer number of consecutive vectors for which transformed data is to be returned, starting at the vector specified at input <4>. This input is only used when the command name at input <2> represents an XFORM VECtor command (not an XFORM MATRIX command). No more than 2048 consecutive vectors may be returned. The default value is 2048.

6. Output <1> contains the transformed data in a format which can only be accepted by input <1> of F:LIST. (F:LIST then prints out the data in ASCII format -- either a PS 300 VECTOR_LIST command or a PS 300 MATRIX_4X4 command, depending on whether the command named at input <2> was an XFORM VECtor or an XFORM MATRIX.)

7. F:XFORMDATA is used in connection with rendering lines and spheres on the PS 340 raster display. This functionality is described in Version A2.V01 of the PS 340 Graphics Firmware Release Notes.

```
                          CSM
                        (CSM2)
                  ___________________
                 |                   |
     B ------->  |<1>             <1>|------> Connected to System
                 |                   |        at initialization
                 |                   |
                 |        C C        |
                 |___________________|
```

PURPOSE

Sets the Color Shadow Mask (CSM) calligraphic display on or off for the Terminal Emulator, for MESSAGE_DISPLAY and for the user's data structures.

DESCRIPTION

INPUT
<1> - TRUE = CSM on, FALSE = CSM off

OUTPUT
<1> - connected to System

DEFAULT

The default is FALSE, setting the CSM off.

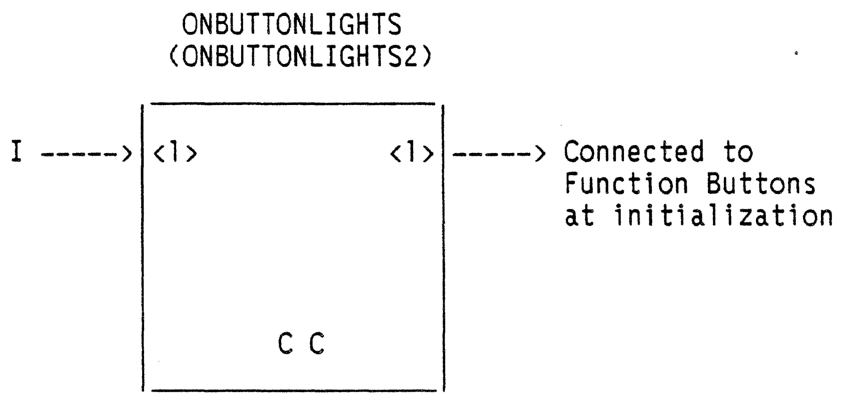
NOTES

1. A TRUE sent to input <1> of CSM slows the speed of the line generator for the CSM calligraphic display. This results in lines that have brighter colors and better end point match.

Initial Function Instance
Output

ONBUTTONLIGHTS

Version A1.V02



PURPOSE

Turns on lighted buttons on the Function Buttons unit.

DESCRIPTION

INPUT
<1> - integer (1 through 32) indicating the button number

OUTPUT
<1> - connected to Function Buttons

NOTES

1. Each button may be turned on independently or all buttons may be turned on by a single message. A zero (0) or any out-of-range integer at input <1> turns on all button lights. An integer from 1 to 32 at input <1> turns on the corresponding button light.

2. Function buttons are arranged in one row of four, four rows of six, and another row of four. They are numbered from left to right starting from the top row. The top row is numbered 1 through 4; the second row 5 through 10, and so on until the last row, 29 through 32.

```
                        PICK
                       (PICK2)
                 ___________________
Any ----------> |<1>             <1>| ------> PL
                |                   |
B ------------> |<2> C           <2>| ------> B
                |                   |
I ------------> |<3> C           <3>| ------> B
                |                   |
                |        D D        |
                |___________________|
```

PURPOSE

Interfaces with the hardware picking circuitry. Any message on input <1> arms the PICK function. Once PICK is enabled, when a pick occurs, the pick list associated with the picked data is sent out on output <1> and a Boolean FALSE is sent on output <2>. Typically, this Boolean is used to disable picking of a set of objects by connecting it to a SET PICKING ON/OFF node in a display tree.

DESCRIPTION

INPUT
<1> - trigger
<2> - TRUE = coordinate, FALSE = index (constant)
<3> - timeout duration (constant)

OUTPUT
<1> - pick list
<2> - FALSE = pick enabled
<3> - FALSE = ACP attempted an unsuccessful pick or timeout occurred

PS 350 Modification

As noted above, output <3> of PICK now reports a FALSE when the ACP attempts a pick and is unsuccessful as well as when the timeout specified on input <3> is exceeded.

```
                          PICK
                        (PICK2)
                 _______________________
Any ----------->|<1>                 <1>|------> PL
                |                       |
B ------------->|<2> C               <2>|------> B
                |                       |
I ------------->|<3> C               <3>|------> B
                |                       |
R ------------->|<4>                    |
                |                       |
I ------------->|<5>                    |
                |                       |
R ------------->|<6>                    |
                |                       |
                |         D D           |
                |_______________________|
```

PURPOSE

Interfaces with the hardware picking circuitry. Any message on input <1> arms the PICK function. Once PICK is enabled, when a pick occurs, the pick list associated with the picked data is sent out on output <1> and a Boolean FALSE is sent out on output <2>. Typically, this Boolean is used to disable picking of a set of objects by connecting it to a SET PICKING ON/OFF node in a display tree.

DESCRIPTION

INPUT
- <1> - trigger
- <2> - TRUE = coordinate, FALSE = index (constant)
- <3> - timeout duration (constant)
- <4> - defines pick window half size for the ACP pass of the pick
- <5> - retry count
- <6> - half-size increment to be added to window half-size on each retry

OUTPUT
- <1> - pick list
- <2> - FALSE = pick enabled
- <3> - FALSE = timeout elapsed

PS 350/PS 390
A2.V02 - April 1987(continued)

NOTES

1. Input <2> selects the kind of pick list that will be output on output <1>. A FALSE on input <2> indicates that the output pick list will be the pick identifier and an index into the vector list or the character string. (The index into the vector list identifies its position in the list; vector 3 is the third vector in a vector list. The index into a character string identifies the picked character by its position in the string; character 5 is the fifth character in a string.)

2. A TRUE on input <2> indicates that the output pick list will include, in addition to the pick identifier and the index, the picked coordinates and the dimension of the picked vector. If the vector is part of a polynomial curve, its parameter value, t, is supplied instead of the index.

3. Coordinate picking on a character string returns an index into the string, not its picked coordinates.

4. Coordinate picking cannot be performed on a vector over 500 [LENGTH] units long.

5. The pick list on output <1> is typically connected to an instance of F:PICKINFO to convert the pick list to a locally useful format. If the pick list is to be printed out, output <1> may be connected to F:PRINT to convert the pick list code to printable characters.

6. When several vectors are picked, the first vector drawn by the Line Generator is reported as picked. For example, if three vectors in a single vector list were picked simultaneously (at a point of intersection), the first vector listed in the object definition would be reported as picked.

7. The integer on input <3> specifies a pick timeout period in refresh frames. This pick timeout period allows the user to determine whether a pick has occurred within the specified amount of time. Timing starts when the PICK function is armed with a message on active input <1>. Allowable integers for input <3> are from 4 through 60.

NOTES (continued)

8. If input <3> is not used, all picks will be reported once the function is armed because no timeout duration has been specified.

9. Typically, the FALSE at output <3> would be used to turn off picking in a display tree (at a SET PICKING ON/OFF node) or to send a "NO PICK" message (probably via F:SYNC(2)) back to the host.

10. The user has three means of cancelling an existing pick timeout duration:

    a. Send an INITialize command. This will remove the PICK function and replace it with a new instance of the PICK function.

    b. Send a non-integer (and ignore the "Bad message" error).

    c. Send an integer less than 4 or greater than 60 to input <3> (and ignore the "Bad message" error).

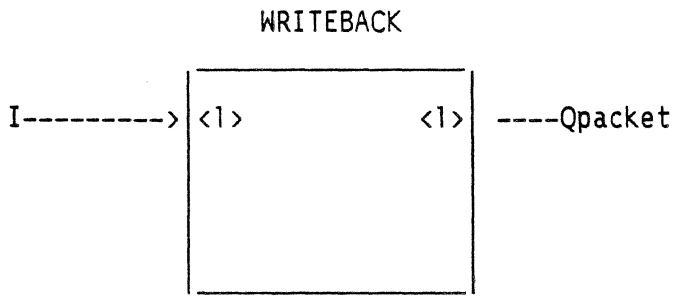
11. Input <4> is a real number between 0 and 1 that defines the pick window half-size for the ACP pass of the pick. This is different from the size set by the SET PICKing LOCation operation node. The Line Generator or the Frame Buffer uses the operation node to determine if a pick has occurred; whereas the ACP uses input <4> to do the actual pick pass on the data.

12. Input <5> is an integer specifying pick pass retries. Since it is possible that the ACP will not find the picked data during a pick pass, input <5> indicates the number of times to add the window increment on input <6> and try another pick pass.

13. Input <6> is a real number between 0 and 1 which specifies the amount to increase the pick window half size on each retry of the pick pass.

EXAMPLE

If a 10 is sent to constant input <3>, then the PICK function is armed with a message on input <1>. The function waits 10 refresh frames from the time the input <1> message is received before checking to see if a pick has occurred. If a pick has occurred within that period, the function outputs the appropriate pick list. If a pick has not occurred, the function outputs a FALSE on output <3>. In either case, the PICK function is disarmed and must be rearmed via input <1> before further picking can be reported.



PURPOSE

WRITEBACK is initialized by the system and is used to send encoded writeback data to user function networks.

This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any WRITEBACK operation node.

DESCRIPTION

INPUT

WRITEBACK has one input queue. Input <1> accepts integers specifying the size of Qpackets to be output by the function. The default size is 512. Minimum and maximum sizes are 16 and 1024. If the size specified on the input is not within this range, the default size will be used.

OUTPUT

WRITEBACK has one output queue. Output <1> passes the encoded writeback data out as Qpackets.

NOTES

WRITEBACK will return all data that are under the WRITEBACK operation node. Host-resident code will be responsible for recognizing the start-of-writeback and end-of-writeback commands. Attribute information, such as color, must be interpreted by host code to ensure that the hardcopy plots are correct.

On the PS 350, viewport translations have not been applied to the data. To correctly compute the position of endpoints, the host program interpreting the writeback code must add a viewport center to each endpoint. The initial viewport center is established with a VIEWPORT CENTER command. The VIEWPORT CENTER command is sent following the start-of-writeback command. Any changes to the viewport center will be indicated through this sequence of commands: CLEAR DDA, CLEAR SAVE POINT, position endpoint, CLEAR SAVE POINT. The position endpoint becomes the new viewport center.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PAttach (   %DESCR Modifiers : P_VaryingType;
                     PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure attaches the PS 300 to the communications channel.

If this procedure is not called prior to use of the Application Procedures, the error code value corresponding to the name PSE_NotAtt is generated, indicating that the PS 300 communications link has not been established.

The parameter (Modify) must contain the phrases:

```
LOGDEVNAM=name/PHYDEVTYP=type
```

where "name" refers to the logical name of the device that the GSRs will communicate with, i.e. TTA6:, TTB2: XME0:, PS:, etc. and "type" refers to the physical device type of the hardware interface that the GSRs will communicate through. This last argument can only be one of the following four interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
PARALLEL (Parallel interface option)
ETHERNET (DECnet Ethernet option)

The parameter string must contain EXACTLY one "/" and blanks are NOT allowed to surround the "=" in the phrases. The PAttach parameter string is not sensitive to upper or lower case.

Example: PAttach ('logdevnam=tta2:/phydevtyp=async', Error_Handler);

where "tta2" is the logical device name of the PS 300, and the hardware interface is standard asynchronous RS-232.

Example: PAttach ('logdevnam=ps:/phydevtyp=dmr-11', Error_Handler);

where the physical device type is a DMR-11 interface, and where the user has informed the VAX that the logical symbol "ps" refers to the name of the logical device that the GSRs will communicate with using the following ASSIGN command:

```
$ ASSIGN XMD0: PS
$ RUN <application-pgm>
```

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PAttch (Modify, ErrHnd)
```

where:

Modify is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine attaches the PS 300 to the communications channel. If this subroutine is not called prior to use of the Application Subroutines, the user's error handler is invoked with the "The PS 300 communications link has not been established" error code corresponding to the mnemonic: PSENOA:.

The parameter (Modify) must contain the phrases:

LOGDEVNAM=name/PHYDEVTYP=type

where "name" refers to the logical name of the device that the GSRs will communicate with, i.e. TTA6:, TTB2: XME0:, PS:, etc. and "type" refers to the physical device type of the hardware interface that the GSRs will communicate through. This last argument can only be one of the following four interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
PARALLEL (high speed parallel interface
ETHERNET (DECnet Ethernet option)

The parameter string must contain EXACTLY 1 "/" and blanks are NOT allowed to surround the "=" in the phrases. The Pattch parameter string is not sensitive to upper or lower case.

Example: CALL PAttch ('logdevnam=tta2:/phydevtyp=async', Errhnd)

where "tta2" is the logical device name of the PS 300, and the hardware interface is standard asynchronous RS-232.

(Continued on next page)

```
Example: CALL PAttch ('logdevnam=ps:/phydevtyp=dmr-11', ErrHnd)
```

where the physical device type is a DMR-11 interface and where the user has informed the VAX that the logical symbol "ps" refers to the name of the logical device that the GSRs will communicate with using the following ASSIGN command:

```
$ ASSIGN XMD0: PS:
$ RUN <application-pgm>
```

UTILITY PROCEDURE AND PARAMETERS

```
[GLOBAL] PROCEDURE PDevInfo (     VAR Channel_num     : INTEGER;
                                  VAR Device_type     : INTEGER;
                                  VAR Dev_status      : INTEGER;
                              PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used to return the Q I/O channel number so that users do not need to detach from the GSRs while doing Physical I/O.

Channel is the VAX Q I/O channel number.

Device is the device code, where:

1 is the code for the DRM-11 interface
2 is the code for the standard asynchronous interface
3 is the code for the Parallel interface

Status is the status where:

0 is not attached
1 is attached

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PDINFO (Channel, Device, Status, ErrHnd)
```

where:

Channel is an INTEGER*4 that is the VAX Q I/O channel number

Device is an INTEGER*4 that is the device code, where:

1 is the code for the DRM-11 interface
2 is the code for the asynchronous interface
3 is the code for the Parallel interface

Status is an INTEGER*4 that is the status where:

0 is not attached
1 is attached

ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine is used to return the Q I/O channel number so that users do not need to detach from the GSRs while doing Physical I/O.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

CALL PVcBeg (Name, VecCou, BNorm, CBlend, Dimen, Class, ErrHnd)

where:

Name is a CHARACTER STRING defining the name of the vector list

VecCou is an INTEGER*4 specifying the total number of vectors in the vector list

BNorm is a LOGICAL*1 defined: .TRUE. for Block Normalized, .FALSE. for Vector Normalized

CBlend is a LOGICAL*1 defined: .TRUE. for Color Blending, .FALSE. for normal depth cueing

Dimen is an INTEGER*4 2 or 3 (2 or 3 dimensions respectively)

*Class is an INTEGER*4 defining the class of the vector list

ErrHnd is the user-defined error-handler subroutine.

This subroutine must be called to begin a vector list. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector-normalized vector lists.)

PVcEnd

Together, the above 3 subroutines implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVCBEG application subroutine. In the PS 300 command, dimension is implied by syntax.

(Continued on next page)

Name := VECTOR_LIST (no corresponding command)

* These mnemonics may be referenced directly by the user if PROCONST.FOR is INCLUDED in the subroutine. See the section on Programming Suggestions for a description of PROCONST.FOR. A description of the vector classes and their INTEGER*4 value is given below.

Mnemonic	Meaning	INTEGER*4 Value
PVCONN	Connected	0
PVDOTS	Dots	1
PVITEM	Itemized	2
PVSEPA	Separate	3
PVTAB	Tabulated	4

Note: If the vector list is class PVTAB, then the BNorm must be FALSE and Dimen must be equal to 3; that is, tabulated vector lists must be vector-normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVcBeg (Name, VecCou, BNorm, CBlend, Dimen, Class, ErrHnd)
```

where:

Name is a CHARACTER STRING defining the name of the vector list

VecCou is an INTEGER*4 specifying the total number of vectors in the vector list

BNorm is a LOGICAL*1 defined: .TRUE. for Block Normalized, .FALSE. for Vector Normalized

CBlend is a LOGICAL*1 defined: .TRUE. for Color Blending, .FALSE. for normal depth cueing

Dimen is an INTEGER*4 2 or 3 (2 or 3 dimensions respectively)

*Class is an INTEGER*4 defining the class of the vector list

ErrHnd is the user-defined error-handler subroutine.

This subroutine must be called to begin a vector list. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector-normalized vector lists)
PVcEnd

Together, the above 3 subroutines implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVCBEG application subroutine. In the PS 300 command, dimension is implied by syntax.

(Continued on next page)

Name := VECTOR_LIST (no corresponding command)

* These mnemonics may be referenced directly by the user if PROCONST.FOR is INCLUDED in the subroutine. See the section on Programming Suggestions for a description of PROCONST.FOR. A description of the vector classes and their INTEGER*4 value is given below.

Mnemonic	Meaning	INTEGER*4 Value
PVCONN	Connected	0
PVDOTS	Dots	1
PVITEM	Itemized	2
PVSEPA	Separate	3
PVTAB	Tabulated	4

Note: If the vector list is class PVTAB, then the BNorm must be FALSE and Dimen must be equal to 3; that is, tabulated vector lists must be vector normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVcLis (NVec, Vecs, PosLin, ErrHnd)
```

where:

NVec is the number of vectors in the vector list and is defined: INTEGER*4

Vecs is the array containing the vectors of the vector list and is defined: REAL*4 (4, NVec)

```
        where:  Vecs(1,n) = vector n x-component
                Vecs(2,n) = vector n y-component
                Vecs(3,n) = vector n z-component
                Vecs(4,n) = vector n intensity (or hue)
                    0 <= Vecs(4,n) <=1  or
                    0 <= Vecs(4,n) <=127 if vector
                       class is tabulated.
```

PosLin is the array containing the move/positive - draw/line information for each vector. PosLin is defined : LOGICAL*1 PosLin(NVec)

If PosLin(n) = .TRUE. then vector n is a draw(line) vector.

If PosLin(n) = .FALSE. then vector n is a move(position) vector.

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to send a piece of a vector list. For vector-normalized vector lists, this subroutine can be called multiple times to send the vector list down in pieces. Multiple calls to this subroutine are not permitted for the block-normalized vector list case, unless the subroutine PVcMax is called first. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector-normalized vector lists)

PVcEnd

(Continued on next page)

The POSLIN Array is always required, however the CLASS specified in PVcBeg determines how it is used. For CONNECTED, DOTS, and SEPARATE, the user need not specify the contents of POSLIN. For ITEMIZED and TABULATED, the user-specified position/line is used.

The fourth position of Vecs is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vecs can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETCB). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVCLIS GSR routine before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

Together, the subroutines PVcBeg, PVcLis, and PVcEnd implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

CALL PVcLis (NVec, Vecs, PosLin, ErrHnd)

where:

NVec is the number of vectors in the vector list and is defined: INTEGER*4

Vecs is the array containing the vectors of the vector list and is defined: REAL*4 (4, NVec)

```
where:  Vecs(1,n) = vector n x-component
        Vecs(2,n) = vector n y-component
        Vecs(3,n) = vector n z-component
        Vecs(4,n) = vector n intensity (or hue)
             0 <= Vecs(4,n) <=1  or
             0 <= Vecs(4,n) <=127 if vector
                class is tabulated.
```

PosLin is the array containing the move/positive - draw/line information for each vector. PosLin is defined : LOGICAL*1 PosLin(NVec)

If PosLin(n) = .TRUE. then vector n is a draw(line) vector.

If PosLin(n) = .FALSE. then vector n is a move(position) vector.

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to send a piece of a vector list. For vector-normalized vector lists, this subroutine can be called multiple times to send the vector list down in pieces. Multiple calls to this subroutine are not permitted for the block-normalized vector list case, unless the subroutine PVcMax is called first. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector normalized vector lists)

PVcEnd

(Continued on next page)

The POSLIN Array is always required, however the CLASS specified in PVcBeg determines how it is used. For CONNECTED, DOTS, and SEPARATE, the user need not specify the contents of POSLIN. For ITEMIZED and TABULATED, the user-specified position/line is used.

The fourth position of Vecs is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vecs can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETCB). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVCLIS GSR routine before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

Together, the subroutines PVcBeg, PVcLis, and PVcEnd implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
SUBROUTINE PVCMAX  (MAX, ERRHAND)
```

DEFINITION

This subroutine must be called before calling PVCLis if creating a creating a block-normalized vector list with multiple calls to PVCLis. To send a vector list, the user must call:

- PVCBeg
- PVCMax (If making calls to PVCLis and creating a block-normalized vector list.)
- PVCLis (This may be called multiple times for vector-normalized vector lists.)
- PVcEnd (This must be last.)

Together, the above 4 procedures implement the PS 300 command

```
Name :=  VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
         N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
SUBROUTINE PVCMAX  (MAX, ERRHAND)
```

DEFINITION

This subroutine must be called before calling PVCLis if creating a creating a block-normalized vector list with multiple calls to PVCLis. To send a vector list, the user must call:

- PVCBeg
- PVCMax (If making calls to PVCLis and creating a block-normalized vector list.)
- PVCLis (This may be called multiple times for vector-normalized vector lists.)
- PVcEnd (This must be last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecBegn (   %DESCR Name              : P_VaryingType;
                              VectorCount       : INTEGER;
                              BlockNormalized   : BOOLEAN;
                              ColorBlending     : BOOLEAN;
                              Dimen             : INTEGER;
                              Class             : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to begin a vector list. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedure may be called multiple times for vector-normalized vector lists)

PVecEnd

It contains the following parametric definitions:

- Name specifies the name to be given to the vector list
- VectorCount is the number of vectors to be created
- BlockNormalized is TRUE for Block Normalized and FALSE for Vector Normalized
- ColorBlending is TRUE for Color Blending and FALSE for normal depth cueing
- Dimen is 2 or 3 (2 or 3 dimensions respectively)
- *Class corresponds to a vector class
- Error_Handler is the user-defined error-handler procedure

(Continued on next page)

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR__LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVECBEGN application procedure. In the PS 300 command, dimension is implied by syntax.

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure.

Mnemonic	Meaning	INTEGER Value
P_Conn	Connected	0
P_Dots	Dots	1
P_Item	Itemized	2
P_Sepa	Separate	3
P_Tab	Tabulated	4

Note: If the vector list is class P_Tab, BlockNormalized must be FALSE, and Dimen must be equal to 3; that is, tabulated vector lists must be vector-normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecBegn (   %DESCR Name             : P_VaryingType;
                              VectorCount      : INTEGER;
                              BlockNormalized  : BOOLEAN;
                              ColorBlending    : BOOLEAN;
                              Dimen            : INTEGER;
                              Class            : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to begin a vector list. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedure may be called multiple times for vector-normalized vector lists)

PVecEnd

It contains the following parametric definitions:

- Name specifies the name to be given to the vector list
- VectorCount is the number of vectors to be created
- BlockNormalized is TRUE for Block Normalized and FALSE for Vector Normalized
- ColorBlending is TRUE for Color Blending and FALSE for normal depth cueing
- Dimen is 2 or 3 (2 or 3 dimensions respectively)
- *Class corresponds to a vector class
- Error_Handler is the user-defined error-handler procedure

(Continued on next page)

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR__LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVECBEGN application procedure. In the PS 300 command, dimension is implied by syntax.

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure.

Mnemonic	Meaning	INTEGER Value
P_Conn	Connected	0
P_Dots	Dots	1
P_Item	Itemized	2
P_Sepa	Separate	3
P_Tab	Tabulated	4

Note: If the vector list is class P_Tab, BlockNormalized must be FALSE, and Dimen must be equal to 3; that is, tabulated vector lists must be vector-normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecList (     NumberOfVectors    : INTEGER;
                         VAR Vectors        : P_VectorListType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to send a piece of a vector list. For vector-normalized vector lists, this procedure can be called repeatedly to send the vector list down in pieces. Multiple calls to this procedure are not permitted for the block-normalized vector list case, unless the procedure PVecMax is called first. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedures may be called multiple times for vector-normalized vector lists)

PVecEnd

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

Vectors is the array containing the vectors of the vector list.

where: Vectors [n].V4[1] := Vector n x-component
Vectors [n].V4[2] := Vector n y-component
Vectors [n].V4[3] := Vector n z-component
Vectors [n].V4[4] := Vector n intensity (or hue)
0 <= vectors [n].V4[4] <=1 or 0<= Vectors[n].V4[4] <=127 if vector class is tabulated.

Vectors [n].Draw := True if vector n is a draw/line vector.
Vectors [n].Draw := False if vector n is a move/position vector.

The fourth position of Vectors is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vectors can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETBLND). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVECLIST GSR procedure before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECTORS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

If specifying P_Conn, P_Dots, or P_Sepa, the vector's draw section of the vector list is generated by the procedure. P_Item and P_Tab require that the move/draw nature of each vector be defined by the user.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecList (      NumberOfVectors    : INTEGER;
                          VAR Vectors        : P_VectorListType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to send a piece of a vector list. For vector-normalized vector lists, this procedure can be called repeatedly to send the vector list down in pieces. Multiple calls to this procedure are not permitted for the block-normalized vector list case, unless the procedure PVecMax is called first. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedures may be called multiple times for vector-normalized vector lists)

PVecEnd

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

Vectors is the array containing the vectors of the vector list.

where: Vectors [n].V4[1] := Vector n x-component
Vectors [n].V4[2] := Vector n y-component
Vectors [n].V4[3] := Vector n z-component
Vectors [n].V4[4] := Vector n intensity (or hue)
0 <= vectors [n].V4[4] <=1 or 0<= Vectors[n].V4[4] <=127 if vector class is tabulated.

Vectors [n].Draw := True if vector n is a draw/line vector.
Vectors [n].Draw := False if vector n is a move/position vector.

The fourth position of Vectors is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vectors can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETBLND). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVECLIST GSR procedure before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECTORS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

If specifying P_Conn, P_Dots, or P_Sepa, the vector's draw section of the vector list is generated by the procedure. P_Item and P_Tab requires that the move/draw nature of each vector be defined by the user.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
[GLOBAL, CHECK(NOBOUNDS)] PROCEDURE PVecMax (Maxcomp : REAL)
          (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to set the maximum component of a vector list for multiple calls to PVecList with block-normalized vectors. To send a vector list, the user must call:

- PVecBegn
- PVecMax (If defining block-normalized vector with multiple calls to PVecList)
- PVecList (This may be called multiple times.)
- PVecEnd (This is called last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecMax (Maxcomp : REAL)
          (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to set the maximum component of a vector list for multiple calls to PVecList with block-normalized vectors. To send a vector list, the user must call:

- PVecBegn
- PVecMax (If defining block normalized-vector with multiple calls to PVecList)
- PVecList (This may be called multiple times.)
- PVecEnd (This is called last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := WRITEBACK

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PWRTBACK (    CONST Name    : STRING;
                        CONST Name1   : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DESCRIPTION

This procedure enables writeback in the data structure Name1. Writeback is triggered by sending a TRUE to the writeback operation node created with this procedure.

PARAMETERS

Name1 – The name of the structure to which writeback is applied.

PS 300 COMMAND AND SYNTAX

```
name := WRITEBACK [APPLied to Name1];
```

Name := WRITEBACK

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PWrtBack (  %DESCR Name     : P_VaryingType;
                      %DESCR Name1    : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure enables writeback in the data structure Name1. Writeback is triggered by sending a TRUE to the writeback operation node created with this procedure.

PARAMETERS

Name1 – The name of the structure to which writeback is applied.

PS 300 COMMAND AND SYNTAX

```
name := WRITEBACK [APPLied to Name1];
```

Name := WRITEBACK

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PWRTBK  ( Name, Name1, Errhnd)
```

where:

Name1 is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

This subroutine enables writeback in the data structure **Name1**. Writeback is triggered by sending a TRUE to the writeback operation node created with this subroutine.

PARAMETERS

Name1 - The name of the structure to which writeback is applied.

PS 300 COMMAND AND SYNTAX

```
name := WRITEBACK  [APPLied to Name1];
```

Name := SET BLINKING ON/OFF

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSEBOF (Name, OnOff, Name1, Errhnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1
Name1 is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

·This procedure turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

OnOff – TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

Name1 – The name of the structure that will be affected by the command.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINKing switch
        [APPLied to name1];
```

Name := SET BLINKING ON/OFF

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetBOnf  (  CONST Name     : STRING;
                             Onoff    : BOOLEAN;
                       CONST Name1    : STRING;
                     Procedure Error_Handler (Err : INTEGER ));
```

DEFINITION

This procedure turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

OnOff – TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

Name1 – The name of the structure that will be affected by the command.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINKing switch [APPLied to name1];
```

Name := SET BLINKING ON/OFF

PS 350 User's Manual

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetBOnf (  %DESCR Name     : P_VaryingType;
                             Onoff    : BOOLEAN;
                      %DESCR Name1    : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

Onoff - TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

Name1 - The name of the structure that will be affected by the command.

PS 300 COMMAND AND SYNTAX

name := SET BLINKing switch [APPLied to name1];

APPLICATION SUBROUTINE AND PARAMETERS

CALL PSELNT (Name, Pattrn, Cont, Namel, Errhnd)

where:

Name is a CHARACTER STRING
Pattrn is an INTEGER*4
Cont is a LOGICAL*1
Namel is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

This subroutine specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter pattrn is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7-bit positions in Pattern when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	--------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

Cont_ - LOGICAL value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If Cont is TRUE, the line texture will continue from one vector to the next through the endpoint. If Cont is FALSE, the line texture will start and stop and the vector endpoints.

Pattrn - An integer between 1 and 127 that specifies the desired line texture. When pattern is less that 1 or greater than 127, solid lines are produced.

Namel - The name of the structure to which the line texture is applied.

DEFAULTS

The default line texture is a solid line

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a patterned, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The **With Pattern** and **curve** commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 300 COMMAND AND SYNTAX

```
name := SET LINe_texture [AROUnd_corners] pattern
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetLinT • ( CONST Name      : STRING;
                         Pattern       : INTEGER;
                         AroundCorners : BOOLEAN;
                       CONST Name1     : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter pattern is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7 bit positions in pattern when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	---------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners – Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at the vector endpoints.

Pattern – An integer between 1 and 127 that specifies the desired line texture. When pattern is less that 1 or greater than 127, solid lines are produced.

Name1 – The name of the structure to which the line texture is applied.

DEFAULTS

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The **With Pattern** and **curve** commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 300 COMMAND AND SYNTAX

```
name := SET LINe_texture [AROUnd_corners] pattern
                         [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetLinT  (  %DESCR Name       : P_VaryingType;
                          Pattern        : INTEGER;
                          AroundCorners  : BOOLEAN;
                       %DESCR Name1      : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter pattern is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7-bit positions in Pattern when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	--------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners – Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at the vector endpoints.

Pattern – An integer between 1 and 127 that specifies the desired line texture. When pattern is less that 1 or greater than 127, solid lines are produced.

Name1 – The name of the structure to which the line texture is applied.

DEFAULTS

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The With Pattern and curve commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 300 COMMAND AND SYNTAX

```
name := SET LINe_texture [AROUnd_corners] pattern
                          [APPLied to name1];
```

Name := SET BLINK RATE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSEBR (Name, Rate, Namel, Errhnd)
```

where:

Name is a CHARACTER STRING
Rate is an INTEGER*4
Namel is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

Rate – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for the number of specified refreshes and off for the specified number of refreshes.

Namel – The name of the structure to which the blinking rate is applied.

NOTE

PS 330 style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINK RATE n
          [APPLied to namel];
```

Name := SET BLINK RATE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSETBR (   CONST Name     : STRING;
                       Blinkrate    : INTEGER;
                     CONST Name1    : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DESCRIPTION

This procedure specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

Blinkrate - An integer designating the duration of the blink in refresh cycles. The blinking data will be on for the specified number of refreshes and off for the specified number of refreshes.

Name1 - The name of the structure to which the blinking rate is applied.

NOTE

PS 330-style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINK RATE n
        [APPLied to name1];
```

Name := SET BLINK RATE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSETBR (   %DESCR Name     : P_VaryingType;
                         Blinkrate   : INTEGER;
                     %DESCR Name1    : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DESCRIPTION

This procedure specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

Blinkrate – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for the specified number of refreshes and off for the specified number of refreshes.

name1 – The name of the structure to which the blinking rate is applied.

NOTE

PS 330-style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINKING RATE n
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE   PViewL (  CONST Name     : STRING;
                            Hmin     : REAL;
                            Hmax     : REAL;
                            Vmin     : REAL;
                            Vmax     : REAL;
                            Imin     : REAL;
                            Imax     : INTEGER;
                      CONST Name1    : STRING;
                    Procedure Error_Handler (Err : INTEGER));;
```

DEFINITION

The PViewL procedure for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standard PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It affects all objects below the node created by the command in the display tree.

PARAMETERS

Hmin,Hmax,Vmin,Vmax – The x and y boundaries of the new viewport. Values must be within the -1 to 1 range.

Imin,Imax – Specifies the minimum and maximum intensities for the viewport. imin is the intensity of lines at the back clipping plane; imax at the front clipping plane. Values must be within the 0 to 1 range.

Name1 – The name of the structure to which the viewport is applied.

PS 300 COMMAND AND SYNTAX

```
name := LOAD VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE   PViewL  ( %DESCR Name      : P_VaryingType;
                             Hmin      : REAL;
                             Hmax      : REAL;
                             Vmin      : REAL;
                             Vmax      : REAL;
                             Imin      : REAL;
                             Imax      : INTEGER;
                      %DESCR Name1     : P_VaryingType;
                     Procedure Error_Handler (Err : INTEGER ));;
```

DEFINITION

The PViewL procedure for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standand PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines.

PARAMETERS

Hmin,Hmax,Vmin,Vmax - The x and y boundaries of the new viewport. Values must be within the -1 to 1 range.

Imin,Imax - Specifies the minimum and maximum intensities for the viewport. imin is the intensity of lines at the back clipping plane; imax at the front clipping plane. Values must be within the 0 to 1 range.

Name1 - The name of the structure to which the viewport is applied.

PS 300 COMMAND AND SYNTAX

```
name := LOAD VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to name1];
```

Name := LOAD VIEWport

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVIEWL  ( Name, Hmin, Hmax, Vmin, Vmax, Imin, Imax, Namel, Errhnd)
```

where:

Hmin, Hmax are REAL*4
Vmin, Vmax are REAL*4
Imin, Imax are REAL*4
Namel is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

The PViewL subroutine for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standand PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines.

PARAMETERS

Hmin,Hmax,Vmin,Vmax – The x and y boundaries of the new viewport. Values must be within the –1 to 1 range.

Imin,Imax – Specifies the minimum and maximum intensities for the viewport. **imin** is the intensity of lines at the back clipping plane; **imax** at the front clipping plane. Values must be within the 0 to 1.

Namel – The name of the structure to which the viewport is applied.

PS 300 COMMAND AND SYNTAX

```
name := LOAD VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to namel];
```

PS 300 WRITEBACK FEATURE

The Writeback feature allows displayed transformed vector data to be sent back to the host. The position of the writeback node in the display structure determines which transformations will be applied to the writeback data. The system-generated writeback node will include all transformations (viewing and modeling). Once the host has received these data, they can be used to generate hardcopy plots or display host-generated raster images. The user is responsible for retrieval and all subsequent processing of data on the host system.

This guide describes how to use the Writeback feature on all members of the PS 300 family of graphics computers. Operational differences among models are specifically noted.

This guide contains:

- A description of the user interface for the Writeback feature. The user interface consists of the WRITEBACK operation node and the WRITEBACK initial function.
- Constraints on the use of the WRITEBACK operation node.
- Descriptions of the WRITEBACK function.
- A list of the commands that may need to be interpreted by host-resident code to filter writeback data retrieved from the PS 300.
- An example of the sequence of data sent back to the host.
- An example of a host program that retrieves, processes, and files writeback data from the PS 350.

Change-pages supporting the Writeback feature are provided in this guide for the Command Summary, the Function Summary and the Graphics Support Routine sections of the PS 300 Document Set.

Writeback User Interface

The Writeback feature is implemented by:

- Creating the WRITEBACK operation node (or using the system-generated writeback node, WB$).
- Activating the WRITEBACK operation node.
- Connecting the WRITEBACK function to a function network.

WRITEBACK Operation Node

When the PS 300 is booted, a WRITEBACK operation node is created. It is named WB$ and is placed above every user-defined display structure. This node can be triggered if an entire displayed picture is to be included in the writeback data. If writeback of only a portion of the picture is desired, the user must place other WRITEBACK nodes appropriately in the display structure.

A user-defined WRITEBACK operation node is created by the command:

Name := WRITEBACK [APPlied to Name1];

The WRITEBACK node has one input. A TRUE sent to input <1> of the WRITEBACK node triggers writeback for the data structure below the node. This trigger is sent by the user, for example:

SEND TRUE TO <1>name;

triggers that WRITEBACK node. Of course the node could be triggered through a function network using a function key, etc.

A WRITEBACK operation node delimits the structure from which the writeback data will be collected. Only the data nodes below the WRITEBACK operation node in the display structure will be transformed, clipped, viewport scaled perspective divided (as delineated by the placement of the WRITEBACK node), and sent back to the host.

NOTE

On the PS 350, viewport translations will not be applied to the data.

WRITEBACK Operation Node Constraints

Only a displayed structure can be enabled for writeback. This means that the WRITEBACK operation node must be traversed by the display processor and therefore must be included in the displayed portion of the structure. The default WRITEBACK node WB$ is displayed as part of every displayed structure. But, if the user creates another WRITEBACK node and if this node is triggered before being displayed, the following error message will result:

E 8 ACP cannot find your operate node

Any number of WRITEBACK nodes can be placed within a structure. However, only one WRITEBACK operation can occur at a time. If more than one node is triggered, the WRITEBACK operations are performed in the order in which the corresponding nodes were triggered.

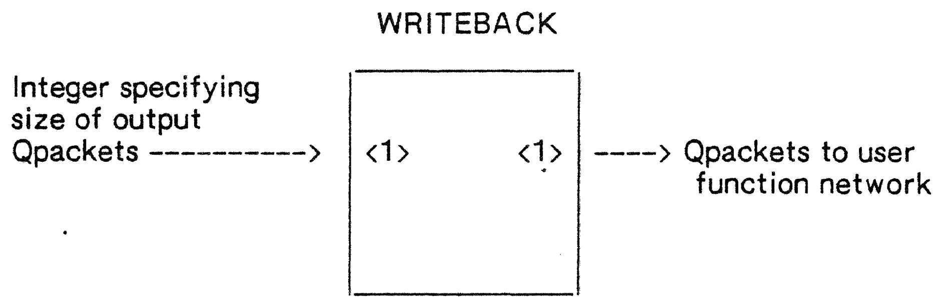
The terminal emulator and message_display information will not be returned to the host.

Polygon data can be returned to the host only if the PS 340 has a 4K ACP.

Before triggering the WRITEBACK operation, disable the SCREENSAVE function by entering the command "SCREENSAVE:= nil;".

The WRITEBACK Function

An initial function instance, WRITEBACK, is created by the system at boot up.


WRITEBACK
Integer specifying size of output Qpackets ---------->
<1>
<1>
----> Qpackets to user function network

WRITEBACK sends encoded writeback data received from the display processor. The writeback data is prefixed by a start-of-writeback command, followed by the encoded data, followed by an end-of-writeback or end-of-frame command.

WRITEBACK has one user-accessible input queue. Input <1> accepts integers specifying the size of Qpackets to be output by the function. The default size is 512 bytes per Qpacket. The minimum and maximum size are 16 bytes per Qpacket and 1024 bytes per Qpacket, respectively. If the size specified by the user is not within this range, the default size will be used by the system.

The input value should be chosen such that the actual size of the qpacket sent to the I/O port is less than or equal to the present input buffer size on the host computer.

If the CVT8TO6 function is used to send the binary data to the host, then the number of the bytes sent to the host is approximately 3/2 * the number of bytes sent by the Writeback function.

For example, if the integer sent to <1> of the Writeback function is 80, the largest Qpacket sent to the host will be 80 * 3/2 = 120. Qpackets, where the size is not a multiple of 4, will be padded to the next multiple of 4. For instance, Qpacket sizes of 77, 78, and 79, sent to CVT8TO6 will all have output sizes of 120.

WRITEBACK has one user-accessible output queue. Output <1> passes the encoded writeback data out as Qpackets until the end-of-writeback or end-of-frame command is seen.

This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any WRITEBACK operation node.

Data Output by WRITEBACK

WRITEBACK will return all data below the WRITEBACK operation node. Host-resident code will be responsible for recognizing the start-of-writeback and end-of-writeback or end-of-frame commands.

Attribute information, such as color, must be interpreted by host code to ensure that the hardcopy plots are correct.

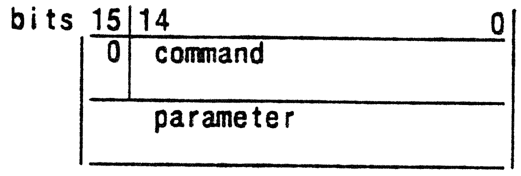
On the PS 350, viewport translations will not be applied to the data. Correct computation of the position of endpoints requires that the host program add a viewport center to each endpoint. The initial viewport center is established with a VIEWPORT CENTER command. The VIEWPORT CENTER command is sent following the start-of-writeback command. Any changes to the viewport center will be indicated through this sequence of commands: CLEAR DDA, CLEAR SAVE POINT, position endpoint, CLEAR SAVE POINT. The position endpoint becomes the new viewport center.

Also, on the PS 350, several commands such as ENABLE PICK and ENABLE BLINK are sent to the host.. These will not typically be needed by the host program. However, these commands come directly from the refresh buffer and are not filtered by the PS 350. Host-resident code must filter the writeback data and strip out nonessential information.

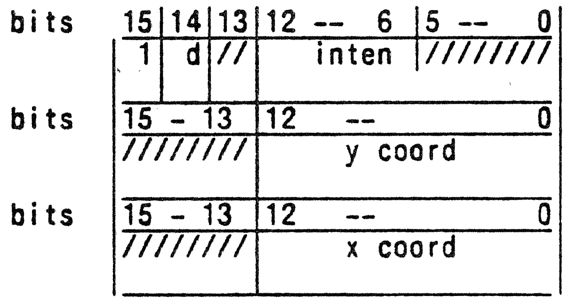
Data Packets Returned

Data packets sent out the WRITEBACK function contain the following information:

- If bit 15 of the first word is 0, it signals that the data that follows is a command. For example, if the first word is H#0200 (Hex 0200) then the Line Generator status will follow.



- If bit 15 of the first word is 1, it indicates that intensity, x and y coordinate information will follow. Intensity can range from 0 to 127. The format of the data is:



if d = 1, then it is a DRAW
if d = 0, it is a MOVE

NOTE

In the illustrations of data format, the slash character is used to illustrate blocks of data that are unused.

Command Descriptions

The following list describes the commands that the host-resident code might have to interpret before it can recognize and filter writeback data received from the PS 300. These commands can be intermixed with vector data.

It is important to note that each command contains at least three 16-bit words. For example, if a command only has one parameter then the third word is unused, but it is still sent to the host. If a command has 3, 4, or 5 parameters, then 6 words will be sent for that command.

START-OF-WRITEBACK code in hex = H#0B00
# 2816

Parameters:
Line texture (one word)
LGS (one word)

Marks the beginning of the writeback segment, of which there is guaranteed to be only one.

The texture and line generator status are included here. They follow the same format as the texture and line generator status shown below.

```
|         B00          |
|/////////| Texture    |
|         LGS          |
```

END-OF-WRITEBACK code in hex = H#0C00
# 3072

Parameters:
None

Marks the end of the writeback segment. For the PS 350, the end-of-writeback may also be indicated by the end-of-frame command.

```
|         C00          |
|    0    |     0/1    |   0 = finished successfully, 1 = cannot finish
|//////////////////////|       operation because of insufficient memory
```

The error code (0 or 1) is currently not present in the PS 350 systems.

LINE GENERATOR STATUS code in hex = H#0200
# 512

Parameters:
Status word (one word)

Indicates dot mode (bit 8) and which display is selected (bits 0-3). Normally, only the dot mode bit must be referenced.

```
|         200          |
|         LGS          |
|//////////////////////|
```

Line Generator Status Register (LGS):

```
|///|///|///|///|///|///|///|SHO|///|///|///////|  SCOPE SELECT  |
|///|///|///|///|///|///|///|EPT|///|///|///////| D   C   B   A  |
 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

Bit	Logical Names	
		B A
08	SHOWENDPT	Dot mode
03	BLANKD	Blank scope D (1 blanks the scope 0 enables the scope)
02	BLANKC	Blank scope C
01	BLANKB	Blank scope B
00	BLANKA	Blank scope A

COLOR

code in hex = H#0400
# 1024

Parameters:
Color value (one word)

```
|         400          |
|   Hue   | Saturation |
|//////////////////////|
```

```
|///                        |///                 ///////////|
|/// HI        HUE       LO |//// HI  SAT     LO ///////////|
 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

TEXTURE

code in hex = H#0500
# 1280

Parameters:
Texture value (one word)

```
|        500          |
|/////////|  Texture  |
| ////////////////////|
```

Line Generator Texture Register:

```
|/////////////////////////////////|    Texture bit pattern     |
|/////////////////////////////////|                            |
 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

H#007F or H#00FF both default to a Solid line.
For non-PS 350 users, the texture will always be H#00FF.

The following commands are for PS 350 users ONLY.

CLEAR DDA — code in hex = H#0100
# 256

Parameters:
None

PICK BOUNDARY — code in hex = H#0300
# 768

Parameters:
Four Boundary Values (4 words)

CLEAR SAVE POINT — code in hex = H#0600
# 1536

Parameters:
None

SET PICK ID — code in hex = H#0700
# 1792

Parameters:
Pick ID Pointer (two words)

SET LightPen MODE — code in hex = H#0800
# 2048

Parameters:
Control Mask (1)
New X,Y (2)
Delta distance (1)
Delta frames (1) (Total five words)

ENABLE PICK — code in hex = H#0900
# 2304

Parameters:
None

DISABLE PICK — code in hex = H#0A00
# 2560

Parameters:
None

SET BLINK RATE

code in hex = H#0D00
3328

Parameters:
Blink Rate (one word)

ENABLE BLINK

code in hex = H#0E00
# 3584

Parameters:
None

DISABLE BLINK

code in hex = H#0F00
# 3840

Parameters:
None

END-OF-FRAME

code in hex = H#1700
# 5888

Parameters:
None

Signifies that the current update cycle is completed and that any following data is part of the next update frame. This also signifies end of the writeback segment.

VIEWPORT CENTER

code in hex = H#1800

Parameters:
x center (one word)
y center (one word)
z center (one word)
spare (two words)

```
  bits 15 ..................... 0
      | coordinates             |  2's complement vector
      |_________________________|
```

This value has to be added to each x,y coordinate pair. This information is necessary to calculate the actual coordinates of the data which has been viewport scaled. Every time a new viewport is traversed by the Arithmetic Control Processor, a new viewport center command will be sent.

NOTE

Codes H#1900 - H#1F00 are reserved for future commands. Code H#0000 is defined as a no-op, and naturally has no parameters.

EXAMPLE OF THE SEQUENCE OF DATA SENT BACK TO THE HOST

The following example illustrates the sequence of data and the data in byte format sent to the host during a WRITEBACK operation.

B00		Start-of-writeback command
//////////	Texture	
LGS		
400		Color command
Hue	Saturation	
////////////////////////		
Intensity		V
Y		E
X		C
:		T
:		O
:		R
:		S
:		
:		
:		
:		
200		Line Generator Status command
LGS		
////////////////////////		
500		Texture command
//////////	Texture	
////////////////////////		
400		Color command
Hue	Saturation	
////////////////////////		
Intensity		V
Y		E
X		C
:		T
:		O
:		R
:		S
:		
:		
:		
:		
:		
C00		End-of-writeback command
	0/1	0 = finished successfully, 1 = cannot
////////////////////////		finish because of insufficient memory

Data in Byte Format

```
0B  00    Start-of-writeback command
00  FF    Texture
04  70    LGS
04  00    Color command
80  00    Hue/Saturation
00  00    Not used
00  FF    Intensity
1Y  FF       Y
1X  FF       X
00  FF    Intensity
2Y  FF       Y
2X  FF       X
 :           :
 :           :
 :           :
02  00    LGS command
04  70    LGS
00  00    Not used
05  00    Texture command
00  FF    Texture
00  00    Not used
04  00    Color command
80  00    Color
00  00    Not used
00  FF    Intensity
1Y  FF       Y
1X  FF       X
 :           :
 :           :
 :           :
0C  00    End-of-writeback command
00  00    Finshed successfully
00  00    Not used.
```

SAMPLE WRITEBACK PROGRAM

```
PROGRAM Writeback(Input,Output,Outfile,Devfile);
{ Program to read writeback data from a PS 350. This program sets up a  }
{ function network to get the writeback data and processes the data and }
{ creates a data file on the host with the data from the PS 350.        }

CONST
 %INCLUDE 'PROCONST.PAS'
 Max_buf = 1024;

TYPE
 Int16 = -32768..32767;
 Max_line = VARYING [Max_buf] OF CHAR;
 %INCLUDE 'PROTYPES.PAS'

VAR
 OUTFILE : TEXT;
 DEVFILE : TEXT;
 DEVSPEC : P_VARYINGTYPE;
 OUTNAME : P_VARYINGTYPE;
 WBNAME  : P_VARYINGTYPE;
 COMMAND : INT16;
 INDEX : INTEGER;
 LEN : INTEGER;
 Inline : P_VARYBUFTYPE;
 vx,vy,vz : REAL;
 In_DDA : BOOLEAN := FALSE;

 %INCLUDE 'PROEXTRN.PAS'

  PROCEDURE ERR (ERROR: INTEGER);
  {}
  { ERROR HANDLER ROUTINE }
  {}
    BEGIN { ERR }
      {}
      WRITELN(' ERROR :=',ERROR);
      HALT;
      {}
    END; { ERR }
  PROCEDURE Setup;
  { Create function network to send writeback data to host }
  { This uses F:cvt8to6 to send 6-bit data to the host }
      BEGIN
      PFnInst('cvt','cvt8',Err);
      Pconnect ('Writeback',1,1,'cvt',Err);
      Pconnect ('cvt',1,1,'host_message', Err);
      PsndStr (CHR(36),2,'cvt',Err);
      PsndFix (48,1,'writeback', Err);
      PNameNil('screensave',Err);
      PPurge( Err);
      END;

{ Utility procedures}
  PROCEDURE Six_to_eight( Inbuf :  Max_line;
   VAR Outbuf : P_VARYBUFTYPE);
  { Data from PS 350 is in six-bit packed format. This procedure unpacks
    data}

  CONST Base = 36;

  TYPE
    Cheat_4 = PACKED RECORD CASE Boolean OF
   TRUE : ( i: UNSIGNED);
   FALSE : ( c: PACKED ARRAY [1..4] OF CHAR);
 END;

  VAR
    w : Cheat_4;
    c_out,cycle_count,buf_index,il,tc : INTEGER;
    first : BOOLEAN;

  BEGIN
    buf_index := 1;
    first := TRUE;
    cycle_count := 1;
    c_out := 4;
    outbuf := '';
    WHILE buf_index <= len DO
      BEGIN
 tc := ORD(Inbuf[buf_index]) - base;
 IF first THEN
   IF tc < 0 THEN
     c_out := 4+tc
   ELSE
     BEGIN
       first := FALSE;
       w.i := tc;
       cycle_count := SUCC(cycle_count);
     END { ELSE tc >= 0 }
```

```
ELSE
  BEGIN
    w.i := w.i * (2**6);
    w.i := UOR(w.i ,tc);
    cycle_count := SUCC(cycle_count);
  END; { ELSE }
IF cycle_count > 6 THEN
  BEGIN
    FOR il := 4 DOWNTO (5-c_out) DO
      Outbuf := outbuf + w.c[il];
    cycle_count := 1;
    first := true;
  END;
buf_index := SUCC(buf_index);
      END; { WHILE }
 END;

 PROCEDURE Next_Block;
 { Get a block of data from the PS 350 and convert from six to eight}
 { bit format }

 VAR Inbuff : Max_line;

 BEGIN
   PGETWAIT(Inbuff,err);
   Index := 1;
   Len := LENGTH(Inbuff);
   Six_to_eight ( Inbuff, Inline);
   Len := LENGTH(Inline);
 END;

 PROCEDURE Get_Value( VAR a : INT16);
 { Convert two bytes of input buffer to 16 bit integer }

 VAR i : INTEGER;

 BEGIN { Get_Value }
   a := 0;
   FOR i := 1 TO 2 DO
     BEGIN
Index := Index + 1;
IF Index > Len THEN
  Next_Block;
a := a * 256 + ORD(Inline[Index]);
     END;
 END;{ Get_Value }
```

```
{ Procedures for processing refresh buffer commands }

  PROCEDURE Clear_DDA;
  { CLEAR DDA - %X0100 }
  { Parameters - None }
  { Indicates start of sequence to set viewport center }
  { This sequence is CLEAR DDA, CLEAR SAVE POINT, Vector, CLEAR SAVE POINT}

  VAR a,b : Int16;

  BEGIN
    In_DDA := TRUE;
    Get_value ( a );
    Get_value ( b );
    Writeln(Outfile,'{Clear DDA}');
  END;

  PROCEDURE Write_LGS;
  { WRITE LINE GENERATOR STATUS - %X0200 }
  { Parameters - Status word (one word)  }
  { Bit   8 : Dot mode.  }
  { Bit   6 : Fast sweep ( Opposite of 7) }
  { Bits  5 -  4: Contrast selection (00-min,11-max)}
  { Bits  3 -  0: Scope select( 1 disables,0 enables)}

  VAR lgs,a : Int16;

  BEGIN
    Get_value ( lgs );
    Get_value ( a );
    Writeln(Outfile,'{Write LGS:',HEX(lgs),'}');
  END;

  PROCEDURE Write_Pick_Bound;
  { WRITE PICK BOUNDARY - %X0300  }
  { Parameters - Left, Right, Bottom, Top }

  VAR l,r,b,t,a : Int16;

  BEGIN
    Get_value ( l );
    Get_value ( r );
    Get_value ( b );
    Get_value ( t );
    Get_value ( a );
    Writeln(Outfile,'{Write_Pick_bound:',HEX(l),HEX(r),HEX(b),HEX(t),'}');
  END;
```

```
PROCEDURE Write_Color;
{ WRITE COLOR - %X0400  }
{ Parameters - Color value (one Word) }
{ Bit  15 : Not Used  }
{ Bits 14 - 8 : Hue (High order in 14)}
{ Bit   7 : Not Used  }
{ Bits  6 - 3 : Sat (High order in 3) }
{ Bits  2 - 0 : Not Used  }

VAR c,a : Int16;

BEGIN
  Get_value ( c );
  Get_value ( a );
  Writeln(Outfile,'{Write_Color:',HEX(c),'}');
END;

PROCEDURE Write_Texture;
{ WRITE TEXTURE - %X0500      }
{ Parameters - Texture value (one word)   }
{ Bits 15 - 7 : Not Used      }
{ Bits  6 - 0 : Texture bit pattern     }

VAR t,a : Int16;

BEGIN
  Get_value ( t );
  Get_value ( a );
  Writeln(Outfile,'{Write_Texture:',HEX(t),'}');
END;

PROCEDURE Clear_Save_Point;
{ CLEAR SAVE POINT - %X0600 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Clear_Save_Point:}');
END;

PROCEDURE Set_Pick_Id;
{ SET PICK ID - %X0700      }
{ Parameters - Pick Id Pointer (two words)}

VAR a,b : Int16;
```

```
BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Set_Pick_Id:',HEX(a),HEX(b),'}');
END;

PROCEDURE Set_Lightpen_Mode;
{ SET LIGHTPEN MODE - %X0800 }
{ Parameters - Control mask     }
{   Tracking cross y   }
{   Tracking cross x   }
{   Delta distance     }
{   Delta frames     }

VAR cm,x,y,dd,df : Int16;

BEGIN
  Get_value ( cm );
  Get_value ( x );
  Get_value ( y );
  Get_value ( dd );
  Get_value ( df );
  Writeln(Outfile,'{Set_Lightpen_mode:',HEX(cm),HEX(x),HEX(y),
    HEX(dd),HEX(df),'}');
END;

PROCEDURE Enable_Pick;
{ ENABLE PICK - %X0900}
{ Parameters - None }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Pick:}');
END;

PROCEDURE Disable_Pick;
{ DISABLE PICK - %X0A00   }
{ Parameters - None     }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Pick:}');
END;
```

```
PROCEDURE Enable_Writeback;
{ ENABLE WRITEBACK - %X0B00 }
{ Parameters - Line Texture }
{   Line Gen Status}

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Writeback:',HEX(a),HEX(b),'}');
END;

PROCEDURE Disable_Writeback;
{ DISABLE WRITEBACK - %X0C00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Writeback:}');
END;

PROCEDURE Set_Blink_Rate;
{ SET BLINK RATE - %X0D00 }
{ Parameters - Blink rate }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Set_Blink_Rate:',HEX(a),'}');
END;

PROCEDURE Enable_Blink;
{ ENABLE BLINK - %X0E00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Blink:}');
END;
```

```
PROCEDURE Disable_Blink;
{ DISABLE BLINK - %X0F00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Blink:}');
END;

PROCEDURE End_Of_Frame;
{ END OF FRAME - %X1700    }
{ Parameters - None      }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{End_Of_Frame:}');
END;

PROCEDURE Viewport_Center;
{ VIEWPORT CENTER - %X1800}
{ Parameters - x center    }
{    y center    }
{    z center    }

VAR xc,yc,zc,a,b : Int16;

BEGIN
  Get_value ( xc );
  Get_value ( yc );
  Get_value ( zc );
  Get_value ( a );
  Get_value ( b );
  vx := xc;
  IF (vx >= 32768) THEN vx := vx - 65536.0;
  vx := vx/32767;
  vy := yc;
  IF (vy >= 32768) THEN vy := vy - 65536.0;
  vy := vy/32767;
  vz := zc;
  IF (vz >= 32768) THEN vz := vz - 65536.0;
  vz := vz/32767;
  Writeln(Outfile,'{Viewport_Center:',vx:6:6,' ',vy:6:6,' ',vz:6:6,'}');
END;
```

```
PROCEDURE Process_Vector;
{ Vector - Bit 15 of command = 1 }
{ Word 1 ( command )   }
{ Bit  15 : Always one for vector }
{ Bit  14 : 1 = Draw, 0 = Move }
{ Bits 12 - 6 : Intensity/2  }
{ Bits  5 - 0 : Not Used  }
{ Word 2 ( y coord)   }
{ Bits 15 - 13: Not Used  }
{ Bits 12 -  0: Y coordinate  }
{ Word 3 ( x coord)   }
{ Bits 15 - 13: Not Used  }
{ Bits 12 -  0: X coordinate  }

VAR a,b : Int16;
  un : UNSIGNED;
  pl : CHAR;
  int,x,y : REAL;

BEGIN
  Get_value ( a );
  Get_value ( b );
  un:=command;
  pl:='l';
  IF (UAND(un,%X4000) = 0) THEN pl := 'p';
  un := UAND(un,%X1FC0);
  int := un;
  IF In_DDA THEN
    vz := int/8128.0
  ELSE
    int := (int/8128.0 + vz) * 2;
  un := a;
  un := UAND(un,%X1FFF);
  y := un;
  IF (y >= %X1000) THEN y := y - %X2000;
  IF In_DDA THEN
    vy := y / %XFFF
  ELSE
    y := y / %XFFF + vy;
  un := b;
  un := UAND(un,%X1FFF);
  x := un;
  IF (x >= %X1000) THEN x := x - %X2000;
  IF In_DDA THEN
    vx := x / %XFFF
  ELSE
    x := x / %XFFF + vx;
  IF In_DDA THEN
    BEGIN
  Writeln(Outfile,'{New View Center:',vx:6:6,' ',vy:6:6,' ',vz:6:6,'}');
  In_DDA := FALSE;
      END
    ELSE
    Writeln(Outfile,'{Vec ',pl,' ',x,',',y,' i=',int,'}');
  END;

  PROCEDURE Unknown;
  VAR a,b : Int16;

  BEGIN
    Get_value ( a );
    Get_value ( b );
    Writeln(Outfile,'{Unknown:',HEX(command),HEX(a),HEX(b),'}');
  END;

BEGIN  { Writeback}
  Write ('Enter Output File Name:');
  Readln(Outname);
  Write ('Enter Writeback Operate Node Name:{WB$ is default mode}');
  Readln(wbname);
  open(Outfile,Outname,new);
  rewrite(Outfile);

  { Look for file specifying line for pattach procedure }
  { Example of record in PSDEV.DAT: }
  { 'logdevnam=tt:/Phydevtyp=async' }
  open(devfile,'psdev',old);
  reset(devfile);
  readln(devfile,devspec);
  close(devfile);

  PATTACH(devspec,err);  { Attach to PS 350 }
  Setup;   { Setup writeback network }

  PNAMENIL('SCREENSAVE', ERR);
  PPURGE(ERR);
  PSndBool(TRUE,1,wbname, Err); { Trigger write back operate }

  Next_block;   { Read in first block of writeback data}

  Index := 0;
  Command := 0;
  vx := 0.0;
  vy := 0.0;
  vz := 0.0;

  { Process writeback buffers until END OF FRAME or END WRITEBACK}
  WHILE (Command <> %X0C00) AND (Command <> %X1700) DO
```

```
    BEGIN
      Get_value(Command);
      IF (Command > 32767) THEN { If bit 15 of command if set}
 Process_vector
   ELSE
   CASE (Command DIV 256) OF
     %X01 : Clear_DDA;
     %X02 : Write_LGS;
     %X03 : Write_Pick_Bound;
     %X04 : Write_Color;
     %X05 : Write_Texture;
     %X06 : Clear_Save_Point;
     %X07 : Set_Pick_Id;
     %X08 : Set_Lightpen_Mode;
     %X09 : Enable_Pick;
     %X0A : Disable_Pick;
     %X0B : Enable_Writeback;
     %X0C : Disable_Writeback;
     %X0D : Set_Blink_Rate;
     %X0E : Enable_Blink;
     %X0F : Disable_Blink;
     %X17 : End_Of_Frame;
     %X18 : Viewport_Center;
     OTHERWISE Unknown;
   END; { CASE }
    END;
  PFNINST('SCREENSAVE', 'SCREENSAVE', ERR  PDETACH(ERR);
  PPURGE(ERR):
  {}
END.  { Writeback}
```

E&S CUSTOMER SERVICE TELEPHONE INFORMATION LIST

Evans & Sutherland Customer Engineering provides a central service numbered staffed by CE representatives who are available to take requests from 9:00 a.m. Eastern Time to 5:00 p.m. Pacific Time (7:00 a.m. to 6:00 p.m. Mountain Time). All calls concerning customer service should be made to one of the following numbers during these hours. Before you call, please have available your customer site number and system tag number. These numbers are on the label attached to your PS 300 display or control unit.

Customers in the continental United States should call toll-free:

1 + 800 + 582-4375

Customers within Utah or outside the continental United States should call Dispatch at:

(801) 582-9412

If problems arise during product installation or you have a question that has not been answered adequately by the customer engineer or the customer service center, contact the regional manager at one of the following Customer Engineering offices:

Eastern Regional Manager
(for Eastern and Central Time Zones)
(518) 885-4639

Western Regional Manager
(for Mountain and Pacific Time Zones)
(916) 448-0355

If the regional office is unable to resolve the problem, you may want to call the appropriate department manager at corporate headquarters:

National Field Operations
(for field service issues)
(801) 582-5847, ext 4843

Software Support
(for sofware issues)
(801) 582-5847, ext 4810

Technical Support
(for hardware issues)
(801) 582-5847, ext 4868

Director of Customer Engineering
(for any unresolved problem)
(801) 582-5847, ext 4840

READER COMMENT FORM **Publication Number** ____________________

Title ____

Your comments will help us provide you with more accurate, complete, and useful documentation. After making your comments in the space below, cut and fold this form as indicated, and tape to secure (please do not staple). This form may be mailed free within the United States. Thank you for your help.

How did you use this publication?

▯ General information	▯ As a reference manual
▯ Guide to operating instructions	▯ Other ____________________

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR
Technical Accuracy Is the manual technically accurate?	▯	▯	▯	▯
Completeness Does the manual contain enough information?	▯	▯	▯	▯
Readability Is the manual easy to read and understand?	▯	▯	▯	▯
Clarity Are the instructions easy to follow?	▯	▯	▯	▯
Organization Is it easy to find needed information?	▯	▯	▯	▯
Illustrations and Examples Are they clear and useful?	▯	▯	▯	▯
Physical Attractiveness What do you think of the overall appearance?	▯	▯	▯	▯

What errors did you find in the manual? (Please include page numbers)____________________

Name ____________________	**Street** ____________________
Title ____________________	**City** ____________________
Department ____________________	**State** ____________________
Company ____________________	**Zip Code** ____________

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 4632 SALT LAKE CITY, UTAH

POSTAGE WILL BE PAID BY ADDRESSEE

EVANS & SUTHERLAND
580 Arapeen Drive
Salt Lake City, Utah 84108

ATTN: IAS TECHNICAL PUBLICATIONS

Fold

Cut along dotted line