PS 300 DOCUMENT SET

VOLUME 3b

PROGRAMMER REFERENCE

Evans & Sutherland assumes no responsibility for errors or inaccuracies in this document. It contains the most complete and accurate information available at the time of publication, and is subject to change without notice.

EVANS & SUTHERLAND COMPUTER CORPORATION
P.O. Box 8700, 580 Arapeen Drive
Salt Lake City, Utah 84121

PS 300 DEC VAX/VMS PASCAL V2 GRAPHICS SUPPORT ROUTINES

USER'S MANUAL

Supported Under PS 300 Graphics Firmware Release A1

Copyright © 1984
EVANS & SUTHERLAND COMPUTER CORPORATION
P.O. Box 8700, 580 Arapeen Drive
Salt Lake City, Utah 84121

CONTENTS

SECTION I

SECTION II

SECTION III

APPENDIX A. SAMPLE PROGRAMS

APPENDIX B. HOST_MESSAGE

INTRODUCTION

The PS 300 VAX PASCAL V2 Graphics Support Routines (GSRs) are a package of Pascal procedures that are executed on the host computer. These procedures allow the host to communicate PS 300 commands directly to the PS 300 Command Interpreter. The GSRs provide procedures for most commands acceptable by the PS 300 Graphics System.

The GSRs described here are written in PASCAL V.2 and are supported only in a VAX/VMS environment.

The purpose of this document is to provide a cross reference between the PS 300 command language and the corresponding Pascal procedures of the GSRs.

This document should be used in conjunction with the **PS 300 Command Summary.** No attempt has been made in this document to provide tutorial information on the use of the PS 300 command language or syntax.

The GSRs are supported under PS 300 Graphics Firmware Release P5.V03 and higher. There are no specific hardware requirements.

This manual is divided into three sections. The first section is a guide to the GSRs. It contains information on the conventions and definitions used in the GSRs. There are several PS 300 commands that have not been implemented in the GSRs. These commands are documented under Excluded Commands.

A section titled Programming Suggestions has been provided that lists the GSR Pascal CONSTant declarations that may be helpful to the user.

An error handling scheme has been employed to catch errors detected by the Graphics Support Routines. A table of the error codes and definitions follows the listing of the Utility and Application Procedures.

The second section of the manual lists each GSR procedure with its corresponding PS 300 command. The procedures are presented in alphabetical order with parameters and the corresponding PS 300 command syntax.

When an example is given, it is shown with both the PS 300 command syntax and the procedure parameters. Any notes following a procedure describe discrepancies or restrictions that apply to the procedure but may not apply to the corresponding PS 300 command.

An alphabetical listing of the PS 300 Commands, the corresponding Pascal procedure, and the appropriate page reference is provided at the front of the description of the procedures.

The third section of the manual contains the appendices. Appendix A contains a sample program that illustrates the use of the GSRs and an example of an error-handling procedure. Appendix B contains a description of the PS 300 system function HOST_MESSAGE. Installation instructions are in the **System Manager Reference**, Volume 5 of this document set.

The (GSRs) were developed at Evans and Sutherland as a standard communication path between the PS 300 and the application program. Prior to this interface, communication with the PS 300 was supported by the Host Resident I/O Routines (PSIO). All commands were sent to the PS 300 as ASCII character strings (with the exception of vector lists). It was the responsibility of the application to format graphical information into the proper PS 300 commands. Typically, this was accomplished using FORTRAN ENCODE/DECODE and FORMAT statements, or equivalents, to build character strings to be sent to the Parser via PSSEND. PSVECS provided a faster communication path by formatting vector data into a "binary" format and including the proper routing information to bypass the Parser and communicate directly with the Command Interpreter.

The GSRs provide a set of procedures that perform all formatting and routing duties for the application. They take advantage of the fact that all data formatting is performed by E&S supported code. The GSRs communicate nearly all commands directly to the Command Interpreter and achieve significant performance improvement over the ASCII form of the commands.

Applications

Typically, the procedures will be used for the following applications:

- Attach to the graphics device
- Create and modify display structures
- Create, connect and modify function networks
- Receive data from the graphics device

Graphics Support Routines Conventions

The Pascal V2 version of the Graphics Support Routines make use of the following program-defined Pascal TYPE definitions.

```
P_VaryingType     = VARYING [P_MaxVaryingSize] OF CHAR;
P_VaryBufType     = VARYING [P_MaxVaryBufSize] OF CHAR;
P_KnotArrayType   = ARRAY [1..P_MaxKnots] OF REAL;
P_MatrixType      = ARRAY [1..4, 1..4] OF REAL
P_VectorType      = RECORD
                    Draw : BOOLEAN;
                    V4   : ARRAY [1..4] OF REAL;
                    END;

P_VectorListType  = ARRAY [1..P_MaxVecListSize] OF P_VectorType
P_PatternType     = ARRAY [1...32] of INTEGER;
```

The Pascal V2 version of the Graphic Support Raster Routines make use of the following program-defined Pascal CONSTANT definitions:

P_MaxRunclrSize = User specified maximum length run color array

```
P_ColorType           = RECORD
    RED   : INTEGER;
    GREEN : INTEGER;
    BLUE  : INTEGER;
End;

P_RunColorType        = RECORD
    COUNT  : INTEGER
    RED    : INTEGER;
    GREEN  : INTEGER;
    BLUE   : INTEGER;
End;

P_RunClrArrayType = ARRAY [1..P_MaxRunclrSize] of P_RunColorType;
```

The following parameters can be changed by the user to any appropriate value WITHOUT having to recompile the GSRs:

```
P_MaxKnots       =  10
P_MaxVecListSize = 200
P_MaxVaryingSize = 255
P_MaxVaryBufSize = 512
```

The procedures are listed by their respective Pascal EXTERNAL declarations. A brief description of the procedures and an explanation of the parameters is given where required.

Utility Procedures

There are two types of supporting procedures. Utility Procedures are specific to the operation of the Graphics Support Routines. These calls are used to attach the PS 300, select multiplexing channels, send and receive messages, and detach.

Application Procedures

The Application Procedures correspond almost one for one with the standard PS 300 Commands. Exceptions and exclusions are given following the text on the Application Procedures.

In most cases, the names for the Application Procedures were derived by choosing an abbreviation of the PS 300 command and prefixing it with a P. Parameter ordering generally coincides with the PS 300 commands as well.

Examples of some of the application procedures are below.

Example 1

For commands which build operate display structures, such as

Name:=operate parameter1,parameter2,..., then apply;

The procedure call is:

Poper('name',parameter1,parameter2,...,'apply', Error_Handler);

where:

oper is an abbreviated form of the PS 300 command such as rotate in x -- Protx

'name' is a character string containing the name to be associated with the operate

parameter1,parameter2,..., are the parameters to be used in computing the operation. These may be booleans, integers, reals, vectors, or matrices.

'apply' is a character string containing the name of the object to which this operate applies.

Error_Handler is the user-defined error-handler procedure.

Example 2

For commands to "send" to functions or display structures, such as

Send datum to <input>dest;

The procedure call is:

PSNDtyp(datum,input,'dest', Error_Handler);

where:

'typ' is an abbreviated form of the PS 300 command such as PSndFiX, PSndM2D,...

datum is what is to be sent. It may be Boolean, integer, real, character string, vector, or matrix.

input is an integer which specifies which input of the destination is being sent to.

'dest' is a character string containing the name of the display structure or function.

Error_Handler is the user-defined error-handler procedure.

Example 3

For commands which create functions and connections such as:

```
Name := f:genfcn;
Name := f:genfcn(n);
Conn name<output>:<input>dest;
DISCONN name<output>:<input>dest;
```

The procedures are:

```
PFNINST    ( 'name', 'genfcn', Error_Handler );
PFNINSTN   ( 'name', 'genfcn', n, Error_Handler );
PCONNECT   ( 'name',output,input,'dest', Error_Handler );
PDISC      ( 'name',output,input,'dest', Error_Handler );
```

where:

'name' is a character string containing the name associated with the function instance.

'genfcn' is a character string containing the name of the system generic function.

n is an integer specifying the number of input/outputs for this function instance.

output,input are integers specifying the output and input numbers.

dest is a character string containing the name of the display data structure

Error_Handler is the user-defined error-handler procedure.

Note that the function names in the GSRs are specified without the "F:" prefix that is used in the standard PS 300 command language.

EXCEPTIONS

There are two PS 300 commands that use three procedures. These are the PS 300 LABEL command and the VECTOR_LIST command. For both these commands, the Graphics Support Routines require three separate calls.

To create, specify and complete a label block, the user must call:

PLabBegn - To create and open a label block

PLabAdd - May be called multiple times to add to a previously opened label block

PLabEnd - To complete the creation of a label block.

Together these three procedures implement the PS 300 command:

```
Name := LABELS x, y, z, 'string'
                  .
                  .
                x, y, z, 'string';
```

In the same way, the user must use PVecBegn to begin a vector list, PVecList to send a piece of a vector list, and PVecEnd to end a vector list.

An example of a procedure that varies slightly from the PS 300 command is PBSPL; the PS 300 BSPLINE command. In the PS 300 command language, some of the parameters are optional. In the procedure they are all required. This is also the case for the PRBSPL, PPOLY, and PRPOLY procedures.

The PS 300 syntax allows for instancing multiple display entities and for creating multiple variables. In the PS 300 command language the commands would be:

```
NAME:= INSTANCE a,b,c,d;
```

for instancing multiple display entities, and

```
VARIABLE s,y,z,w,t,q;
```

for multiple variables.

To perform the equivalent instancing of multiple display entities or for creating multiple variables, the following GSR procedures should be used.

For the multiple instance case:

```
PINST('NAME', 'A', Error_Handler);
PINCL('B', 'NAME', Error_Handler);
PINCL('C', 'NAME', Error_Handler);
PINCL('D', 'NAME', Error_Handler);
```

For the multiple variable case:

```
PVAR ('S', Error_Handler);
PVAR ('Y', Error_Handler);
PVAR ('Z', Error_Handler);
PVAR ('W', Error_Handler);
PVAR ('T', Error_Handler);
PVAR ('Q', Error_Handler);
```

EXCLUDED COMMANDS

There are several classes of commands that were not implemented in the Graphics Support Routines. These include unit commands, commands that are currently being reworked in the PS 300 Graphics Firmware, commands that duplicate functionality, and commands that report the status or the configuration of the PS 300.

Units are handled exclusively by the Parser, and as such cannot be passed as binary data to the Command Interpreter. Commands that are currently being reworked in the firmware will be added to the Graphics Support Routines at a later date. The command status and system configuration commands have no applications in an interactive program.

A list of the excluded commands and the reason for their exclusion is shown in the following table.

TABLE 1

COMMAND	REASON FOR EXCLUSION
Begin_Font;	Currently being reworked
End_Font;	Currently being reworked
Store;	Duplicated functionality (use SEND TO)
Look From;	Duplicated functionality (use Look AT)
Command Status	Status command
Setup/Show Interface	System configuration command

Except for the exclusions mentioned above, each PS 300 command corresponds to one or more procedures in the Graphics Support Routines. Commands not implemented in the GSRs are sendable via the PPUTP procedure which sends the command to the PS 300 Parser.

ERROR HANDLING

An error handling scheme has been employed to catch errors detected by the Graphics Support Routines. Examples of errors detected by the Graphics Support Routines are:

Prefix not followed by an operate.
Follow not followed by an operate.
Multiple calls to PVecList for block normalized vector list data.
Invalid characters in a name.

Command Interpreter errors and warnings are not detected by the Graphics Support Routines. Examples of these errors are:

Destination does not yet exist.
Message rejected by destination.
Connection not made.

Error checking will be performed within the GSRs to insure that only valid characters are sent within names, and that procedures are called in the proper order, in cases where order is required. No attempt has been made to capture errors and/or warnings from the Command Interpreter.

Each procedure call includes an argument that specifies the user written error handler. This error handler is of the form:

```
PROCEDURE Error_Handler (Error : INTEGER);
```

where ERROR is an integer error code corresponding to one of the errors.

It is the responsibility of the user to provide an error-handling scheme to decide what action should be taken when an error is detected. The GSRs do not attempt to terminate execution or log errors.

A sample error-handling procedure appears in both of the program examples in Appendix A of this manual. It is a sophisticated error handler that may be incorporated by the user into an error-handling scheme, or used as an example of what an error handler should look like.

The name, description, and error code of each detectable error is given in tables following the description of the Utility and Application Procedures.

EXAMPLES OF THE PROCEDURES

The following two examples show how the procedures are described in this manual.

EXAMPLE - 1

PS 300 DEC VAX/VMS PASCAL GSR PROTX

Name:= ROTATE in X

APPLICATION PROCEDURE AND PARAMETER

```
PROCEDURE PRotX ( %DESCR Name      : P_VaryingType;
                        Angle      : REAL;
                  %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a 3x3 rotation matrix that rotates an object (AppliedTo) around the x axis relative to world space origin. It has the following parametric definitions:

- Angle is the rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in X Angle (APPLied to Apply);

To use the PROTX call, instead of sending the ASCII command string:

```
xrot := ROTate in X 37 applied to object;
```

the application program would call the X-rotation procedure:

```
PRotX ('xrot', 37, 'object', Error_Handler);
```

where 'xrot' is the name of the display structure, 37 is the angle of X rotation, 'object' is the display structure to which the X rotation is to be applied, and the Error_Handler is the user-defined procedure that handles errors detected by the Graphics Support Routines.

The ROTATE IN X example is fairly straight forward, as are the majority of the procedures.

The description of the PCONNECT procedure and its parameters is given in the following example.

EXAMPLE - 2

PS 300 DEC VAX/VMS PASCAL GSR PCONNECT

Name:= CONNECT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PConnect (  %DESCR Source   : P_VaryingType;
                      Out      : INTEGER;
                      Inp      : INTEGER;
                    %DESCR Dest     : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure connects the output (Out) of the function instance (Source) to the input (Inp) of the function instance or display data structure (Dest).

PS 300 COMMAND AND SYNTAX

CONNECT Source <Out>:<Inp> Dest;

Continuing this example, we connect 'name' to the display structure 'xrot' using PConnect as follows.

```
PConnect ('name', 1, 1, 'xrot', Error_Handler);
```

where output <1> of 'name' is connected to input <1> of the display structure 'xrot'.

The PS 300 command syntax for this same operation is:

```
CONNECT name<1>:<1>xrot;
```

PROGRAMMING SUGGESTIONS

The file PROCONST.PAS contains definitions for constants used by the Graphics Support Routines. It is often convenient to think of these constants by name rather than by remembering numbers. Specifically, in the usual PS 300 command syntax, inputs to display structures are often referred to by name such as <append> and <clear> for vector lists and <position> and <step> for character strings. There are also <delete>, <last>, and others. Other useful constants such as values for conditional tests for level of detail, and vector list class are obtainable from PROCONST.PAS.

PROCONST.PAS also contains a complete set of error/warning code definitions. These values may be referenced by name by the user procedure if PROCONST.PAS is INCLUDED in the procedure. The Error Tables in the final section of this manual provide a list of the mnemonics and error codes. Using the mnemonics provides an easy way of checking for the correct error code value.

There are two other files that must be INCLUDED by the user. These additional files and their descriptions are:

PROTYPES.PAS - contains the GSR Pascal TYPE definitions
PROEXTRN.PAS - contains the VAX GSR EXTERNAL Procedure Definitions

The following is an abbreviated list derived from PROCONST.PAS of the constants which should be most useful to the user.

GSR private constant declarations:

Name	Value	Meaning
P_Append	= 0;	<Append> input number.
P_Delete	= -1;	<Delete> input number.
P_Clear	= -2;	<Clear> input number.
P_Step	= -3;	<Step> input number.
P_Position	= -4;	<Position> input number.
P_Last	= -5;	<Last> input number.
P_Substitute	= -6;	<Substitute> input number.
P_LES	= 0;	"Less" level of detail comparison operator.
P_EQL	= 1;	"Equal" level of detail comparison operator.
P_LEQL	= 2;	"Less-equal" level of detail comparison operator.
P_GTR	= 3;	"Greater" level of detail comparison operator.
P_NEQL	= 4;	"Not-equal" level of detail comparison operator.
P_GEQL	= 5;	"Greater-equal" level of detail comparison operator.
P_Conn	= 0;	Vector list "Connected" class type.
P_Dots	= 1	Vector List "Dots" class type.
P_Item	= 2;	Vector List "Itemized" class type.
P_Sepa	= 3;	Vector List "Separate" class type.

INDEX TO THE PROCEDURES

The following list from left to right gives an alphabetical listing of the PS 300 Command Name and the Pascal Application Procedure Name in this manual where the procedure is listed with its parameters.

PS 300 COMMAND NAME	APPLICATION PROCEDURE	PAGE
DISCONNECT ALL	PDISCALL	49
DISCONNECT ‹OUT›	PDISCOUT	50
DISPLAY	PDISPLAY	51
ENABLE/DISABLE RASTER VIDEO	PRASVI	100
END	PEND	52
END OPTIMIZE	PENDOPT	54
END_S	PENDS	53
ERASE PATTERN FROM	PERAPATT	55
ERASE_RASTER_SCREEN	PRASER	96
EYE_BACK	PEYEBACK	56
F:FUNCTION NAME	PFNINST	58
F:FUNCTION NAME (INOUTS)	PFNINSTN	59
FIELD_OF_VIEW	PFOV	63
FOLLOW WITH	PFOLL	60
FORGET	PFORGET	62
IF CONDITIONAL_BIT	PIFBIT	64
IF LEVEL_OF_DETAIL	PIFLEVEL	65
IF PHASE	PIFPHASE	66
ILLUMINATION	PILLUMIN	67
INCLUDE	PINCL	68
INCREMENT LEVEL_OF_DETAIL	PINCLOD	69
INITIALIZE	PINIT	70
INITIALIZE CONNECTIONS	PINITC	71
INITIALIZE DISPLAYS	PINITD	72
INITIALIZE NAMES	PINITN	73

PS 300 COMMAND NAME	APPLICATION PROCEDURE	PAGE
INSTANCE OF	PINST	74
LABELS	PLABADD	75
	PLABBEGN	76
	PLABEND	77
LOAD PIXEL VALUE	PRASWP	101
LOOK AT FROM	PLOOKAT	78
MATRIX_2X2	PMAT2X2	79
MATRIX_3X3	PMAT3X3	80
MATRIX_4X3	PMAT4X3	81
MATRIX_4X4	PMAT4X4	82
NAME := NIL	PNAMENIL	83
OPTIMIZE STRUCTURE	POPTSTRU	84
PATTERN	PDEFPATT	45
PATTERN Name1 WITH Name2	PPATWITH	85
POLYGON (ATTRIBUTES)	PPLYGATR	86
POLYGON (BEGIN)	PPLYGBEG	87
POLYGON (END)	PPLYGEND	88
POLYGON (LIST)	PPLYGLIS	89
POLYGON (OUTLINE)	PPLYGOTL	91
POLYNOMIAL	PPOLY	92
PREFIX NAME WITH	PPREF	94
RATIONAL BSPLINE	PRBSPL	103
RATIONAL POLYNOMIAL	PRPOLY	112
RAWBLOCK	PRAWBLOC	102
REMOVE FOLLOWER OF NAME	PREMFOLL	106
REMOVE FROM	PREMFROM	107

PS 300 COMMAND NAME	APPLICATION PROCEDURE	PAGE
REMOVE NAME	PREM	105
REMOVE PREFIX	PREMPREF	108
RESERVE_WORKING_STORAGE	PRSVSTOR	114
ROTATE IN X	PROTX	109
ROTATE IN Y	PROTY	110
ROTATE IN Z	PROTZ	111
SCALE	PSCALEBY	115
SECTIONING_PLANE	PSECPLAN	116
SEND 2D MATRIX TO	PSNDM2D	139
SEND 2D VECTOR TO	PSNDV2D	145
SEND 3D MATRIX TO	PSNDM3D	140
SEND 3D VECTOR TO	PSNDV3D	146
SEND 4D MATRIX TO	PSNDM4D	141
SEND 4D VECTOR TO	PSNDV4D	147
SEND BOOLEAN TO	PSNDBOOL	137
SEND COUNT*DRAWMV TO	PSNDPL	142
SEND FIX TO	PSNDFIX	138
SEND REAL_NUMBER TO	PSNDREAL	143
SEND STRING TO	PSNDSTR	144
SEND VALUE TO	PSNDVAL	148
SEND VECTOR LIST	PSNDVL	149
SET CHARACTERS SCREEN_ORIENTED	PSETCHRS	120
SET CHARACTERS SCREEN_ORIENTED/FIXED	PSETCHRF	119
SET CHARACTERS WORLD_ORIENTED	PSETCHRW	121
SET COLOR	PSETCOLR	123

PS 300 COMMAND NAME	APPLICATION PROCEDURE	PAGE
SET COLOR BLENDING	PSETBLND	118
SET CONDITIONAL_BIT	PSETBIT	117
SET CONTRAST	PSETCONT	124
SET CSM	PSETCSM	125
SET DEPTH_CLIPPING	PSETDCL	127
SET DISPLAY	PSETDONF	128
SET DISPLAYS ALL	PSETDALL	126
SET INTENSITY	PSETINT	129
SET LEVEL_OF_DETAIL	PSETLOD	130
SET LOGICAL DEVICE COORDINATES	PRASLD	97
SET LOOK_UP_TABLE_RANGE	PRASLR	98
SET PICKING INDENTIFIER	PSETPID	131
SET PICKING LOCATION	PSETPLOC	132
SET PICKING SWITCH	PSETPONF	134
SET PIXEL LOCATION	PRASCP	95
SET PLOTTER	PSETPLOT	133
SET RATE	PSETR	135
SET RATE EXTERNAL	PSETREXT	136
SETUP CNESS	PSETCNES	122
SOLID_RENDERING	PSOLREND	150
STANDARD FONT	PSTDFONT	152
SURFACE_RENDERING	PSURREND	151
TRANSLATE	PTRANSBY	153
VARIABLE NAME	PVAR	154

PS 300 COMMAND NAME	APPLICATION PROCEDURE	PAGE
VECTOR_LIST	PVECBEGN	155
	PVECEND	157
	PVECLIST	158
VIEWPORT	PVIEWP	159
WINDOW	PWINDOW	160
WRITE LOOK_UP_TABLE ENTRIES	PRASLU	99
XFORM MATRIX	PXFMATRX	162
XFORM VECTOR_LIST	PXFVECTR	163

The following is a list of the Utility Procedures.

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: (The PS 300 communications link has not been established) is generated.

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 three interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
DMR-11 (DMR-11 high speed interface)
PARALLEL (Parallel interface option)

The parameter string must contain EXACTLY one "/" somewhere between the above phrases. 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 PROCEDURE AND PARAMETERS

```
PROCEDURE PDetach (PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure detaches (disconnects) the communications link established between the host and the PS 300. This procedure should always be the LAST GSR procedure invoked by an application program.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PGet (   %DESCR Str : P_VaryBufType;
                   PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

The PGet procedure is used to poll the PS 300 for input records by requesting a message that has been sent to the PS 300 function HOST_MESSAGE. The actual message contents and number of bytes read from the PS 300 are returned in: Str.

WARNING

The parameter (Str) MUST be declared to be a P_VaryBufType.

If a PGet call is issued and no message exists to be sent back to the host, then the returned length of the message is 0. Otherwise, the length of the message is greater than 0, and indicates the true number of bytes in the message.

NOTE

If the default value for input <2> or input <3> of HOST_MESSAGEB is changed by the user to be something other than a single carriage return, then the above description no longer applies. The user should refer to Appendix B of this manual for a description of HOST_MESSAGEB and its inputs.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PGetWait (   %DESCR Str : P_VaryBufType;
                       PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

The PGetWait procedure is used to query the PS 300 for input records by requesting a message that has been sent to the PS 300 function HOST_MESSAGE. If no message exists to be read, the PGetWait procedure will wait until a message arrives from HOST_MESSAGE. The actual message contents and number of bytes read are returned in: Str.

WARNING

The parameter (Str) MUST be declared to be a P_VaryBufType.

NOTE

If the default value for input <2> of HOST_MESSAGEB is changed by the user to be something other than a single carriage return, the above description no longer applies. The user should refer to Appendix B of this manual for a description of the function and its inputs.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PMuxCI (          NewCIChan : INTEGER;
                  PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure defines a new CIROUTE output channel to be accessed as the Binary CI channel. The standard and default CI channel is 2.

The parameter for NewCIChan is an INTEGER that represents the actual output channel to be accessed as the Binary CI channel.

This procedure is provided to allow for a PS 300 to be configured with multiple Command Interpreters.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PMuxG (          NewMuxChan : INTEGER;
                  PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

The procedure defines the CIROUTE output channel being currently accessed as the "generic" channel by PPutG. The call is provided to support the future implementation of custom user-functions connected to various outputs of CIROUTE.

The parameter for NewMuxChan is the new CI output channel to be used by PPutG. Examples are shown below:

MuxChn = 1: Send to parser. CIROUTE<3>
MuxChn = 2: Send to READSTREAM CIROUTE<4>
etc.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PMuxPars (          NewParseChan : INTEGER;
                    PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure defines the CIROUTE output channel to be accessed by PPutPars. This procedure allows for the implementation and support of multiple Parsers. The standard and default Parser channel is 1.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PPurge (PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

The GSRs always buffer the output to the PS 300 to achieve maximum I/O efficiency. PPurge allows the user to explicitly purge the output buffer.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PPutG (   %DESCR Str : P_VaryingType;
                    PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure sends the bytes specified in the buffer: Str to the current generic demultiplexing channel of CIROUTE established by: PMuxG.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PPutPars (   %DESCR Str : P_VaryingType;
                       PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure sends the ASCII characters specified in the buffer: Str to the PS 300 parser.

ALLOCATE PLOTTER

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PALLPLOT ( N          : INTERGER;
                     PROCEDURE Error_Handler (Err : INTERGER));
```

DEFINITION

This procedure allocates the plotter specified in PLOT to the calling user. When the plotter is allocated, formfeed after plot is disabled.

PS 300 COMMAND AND SYNTAX

ALLOCATE PLOTTER Plot;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB ( %DESCR Name       : P_Varying Type;
                           Hue        : REAL;
                           Saturation : REAL;
                           Intensity  : REAL;
                           Reserved   : REAL;
                           Diffused   : REAL;          {default .75}
                           Specular   : REAL;          {default 4}
                    Procedure Error_Handler (Err : INTERGER));;
```

DEFINITION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. Hue, Saturation, and Intensity define the color of the polygon. Hue specifies an angle between 0 and 360 indicating the color on a color wheel with full blue begin 0, red being 120 and green being 240. Saturation specifies the saturation of the color with 0 being no color and 1 being full saturation. Intensity specifies the intensity of the color with 0 being no color (black) and 1 being full intenstiy. Diffused is the proportion of color contributed by defuse reflection versus that contributed by specular reflection with a value of 1 eliminating all specular highlighting and a value of 0 eliminating all diffuse reflectivity. Specular adjusts the concentration of specular highlights in the range of 0 to 10.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR Hue[,Sat[Intens]]]
                   [DIFFUSE Diffus]
                   [SPECULAR Specul];
```

Name := ATTRIBUTES ... AND

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB2 ( %DESCR Name         : P_Varying Type;
                            Hue1         : REAL;
                            Saturation1  : REAL;
                            Intensity1   : REAL;
                            Reserved1    : REAL;
                            Diffused1    : REAL;          {default .75}
                            Specular1    : REAL;          {default 4}
                            Hue2         : REAL;
                            Saturation2  : REAL;
                            Intensity2   : REAL;
                            Reserved2    : REAL;
                            Diffused2    : REAL;          {default .75}
                            Specular2    : REAL;          {default 4}
                      Procedure Error_Handler (Err : INTERGER));;
```

DEFINITION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. This is similar to the PATTR procedure but allows for a second set of artributes to be defined for the back side of polygons.

PS 300 COMMAND AND SYNTAX

```
Name :=  ATTRIBUTES  [COLOR Hue[,Sat[Intens]]]
                     [DIFFUSE Diffus]
                     [SPECULAR Specul];
         AND         [COLOR Hue2[,Sat2[,Inten2]]]
                     [DIFFUSE Diffu2]
                     [SPECULAR Specu2];
```

Name := BEGIN

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PBegin (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This call is used with the PEND procedure to group a set of viewing and/or modeling commands so that they appear to be executed simultaneously.

PS 300 COMMAND AND SYNTAX

Name := BEGIN

Name := BEGIN_S

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure is used with the PENDS procedure to group a set of viewing and/or modeling commands so that each element does not need to be explicitly named to be accessed.

PS 300 COMMAND AND SYNTAX

Name := BEGIN_Structure

Name := BSPLINE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PBspl (   %DESCR Name       : P_VaryingType;
                           Order      : INTEGER;
                           OpenClosed : BOOLEAN;
                           NonPer_Per : BOOLEAN;
                           Dim        : INTEGER;
                           N_Vertices : INTEGER;
                       VAR Vertices   : P_VectorListType;
                           KnotCount  : INTEGER;
                       VAR Knots      : P_KnotArrayType;
                           Chords     : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure evaluates a B-spline curve, allowing the parametric description of the curve form without having to specify the coordinates of each vector. In the parametric definitions:

- Name specifies the name to be assigned to the computed B-spline
- Order is the order of the curve
- OpenClosed is TRUE for Open and FALSE for Closed
- NonPer_Per is TRUE for Non/periodic and FALSE for Periodic
- Dim is 2 or 3 (2 or 3 dimensions respectively)
- N-Vertices specifies the number of vertices
- Vertices specifies the vertices of the B-spline
- KnotCount specifies the number of knots
- Knots specifies the knot sequence to be used in computing the B-spline
- Chords is the number of vectors to be created
- Error-handler is the user-defined error handler procedure

(Continued on next page)

Name := BSPLINE

(continued)

PS 300 COMMAND AND SYNTAX

```
Name := BSPLINE
        ORDER = Order
        OPEN/CLOSED
        NONPERIODIC/PERIODIC
        N = NVert
        VERTICES   = X(1),  Y(1),  ( Z(1) )
                     X(2),  Y(2),  ( Z(2) )
                      :      :       :|
                     X(N),  Y(N),  ( Z(N) )
        KNOTS = Knots (1), ... Knots (KntCnt)
        CHORDS = Chords;
```

NOTE

None of the parameters in the application procedure PBSPL are optional. The dimension must be specified in the PBSPL application procedure. In the PS 300 command, dimension is implied by syntax.

If KnotCount = 0, then the default knot sequence is generated and the knots array is ignored.

Name := CHARACTER ROTATE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PCharRot (   %DESCR Name      : P_VaryingType;
                              Angle     : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure rotates the specified characters (AppliedTo) and has the following parametric definition:

- Angle is the Z-rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := CHARacter ROTate Angle (APPLied to AppliedTo);

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PChars (   %DESCR Name      : P_VaryingType;
                            TranX     : REAL;
                            TranY     : REAL;
                            TranZ     : REAL;
                            StepX     : REAL;
                            StepY     : REAL;
                     %DESCR Chars     : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines a character string (Chars) and specifies its location and placement. It has the following parametric definitions:

- TranX, TranY, TranZ give the x,y,z coordinates of the location of the beginning of the character string
- StepX, StepY give the spacing between characters in character unit size

PS 300 COMMAND AND SYNTAX

Name := CHARacters TranX,TranY,TranZ STEP StepX,StepY 'Chars';

Name := CHARACTER SCALE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PCharSca (   %DESCR Name      : P_VaryingType;
                              ScaleX    : REAL;
                              ScaleY    : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a uniform 2x2 scale matrix to scale the specified characters (AppliedTo). It has the following parametric definition:

- ScaleX, ScaleY give the scaling factors for the x,y axes

PS 300 COMMAND AND SYNTAX

Name := CHARacter SCALE ScaleX, ScaleY (APPLied to AppliedTo);

Name := CONNECT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PConnect (   %DESCR Source     : P_VaryingType;
                              Out        : INTEGER;
                              Inp        : INTEGER;
                       %DESCR Dest       : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure connects the output (Out) of the function instance (Source) to the input (Inp) of the function instance or display data structure (Dest).

PS 300 COMMAND AND SYNTAX

CONNECT Source <Out>:<Inp> Dest;

Name := COPY

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PCopyVec (   %DESCR Name      : P_VaryingType;
                       %DESCR CopyFrom  : P_VaryingType;
                              Start     : INTEGER;
                              Count     : INTEGER;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a vector list (Name) containing a group of consecutive vectors copied from another vector list (CopyFrom) where 'Start' is the first vector to be copied and 'Count' is the number of vectors to be copied.

PS 300 COMMAND AND SYNTAX

Name := COPY CopyFrom (START=) Start (,) (COUNT=) Count;

Name := DEALLOCATE PLOTTER

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PDALLPLT (      N  : INTEGER;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure deallocates a plotter previously allocated to the calling user.

PS 300 COMMAND AND SYNTAX

DEALLOCATE PLOTTER Plot;

Name := DECREMENT LEVEL_OF_DETAIL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PDecLOD (   %DESCR Name      : P_VaryingType;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure decrements the current level of detail by 1.

PS 300 COMMAND AND SYNTAX

Name := DECrement LEVel_of_detail (APPLied to AppliedTo);

Name := PATTERN

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PDEFPATT ( %DESCR Name 1          : P_VaryingType;
                            Segments        : INTEGER;
                        VAR Pattern         : P_PatternType;
                            Continuous      : BOOLEAN;
                            Match           : BOOLEAN;
                            Length          : REAL;
                      PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure defines a pattern that can be used to pattern a vector list or curve. Segs defines the number of integers used to define the pattern, those integers given by pattrn. Contin tells whether or not patterning is to go across multiple vectors. Match tells if the pattern length is to be adjusted to make the patterning terminate precisely at the endpoints. Length gives the pattern length.

PS 300 COMMAND AND SYNTAX

```
Name := PATTERN Pattrn [Pattrn(2)...Pattrn(Segs)]
                [AROUND_CORNERS]   [MATCH/NOMATCH]   LENGTH
                Length;
```

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure deletes any previously defined data structure name (Name). After a PDelete call is issued, all functions and data structures referring to (Name) will no longer include the data that was associated with (Name).

PS 300 COMMAND AND SYNTAX

DELete Name;

DEL NAME*

APPLICATION PROCEDURE AND PARAMETERS

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

DESCRIPTION

This procedure deletes all names that begin with the characters specified in the parameter Name.

PS 300 COMMAND AND SYNTAX

DELETE Name*;

DISCONNECT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PDisc (   %DESCR Source     : P_VaryingType;
                           Out        : INTEGER;
                           Inp        : INTEGER;
                    %DESCR Dest       : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure disconnects the specified output number (Out) of function instance (Source) from the input (Inp) of the function instance or display data structure (Dest).

PS 300 COMMAND AND SYNTAX

DISCONNect Source <Out>:<Inp> Dest;

DISCONNECT Source:ALL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PDiscAll (   %DESCR Source     : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure disconnects all outputs of (Source) from all inputs to function instances or display data structures that it was previously connected to.

PS 300 COMMAND AND SYNTAX

DISCONNect Source:ALL;

DISCONNECT <OUT>

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PDiscOut (   %DESCR Source     : P_VaryingType;
                       Out        : INTEGER;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure disconnects the output (Out) of the function instance (Source) from all inputs to function instances or display data structures that it was previously connected to.

PS 300 COMMAND AND SYNTAX

DISCONNect Source <Out>:ALL;

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure displays a data structure (Name).

PS 300 COMMAND AND SYNTAX

DISPlay Name;

END

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used with the PBEGIN procedure to group a set of viewing and/or modeling commands so that they appear to be executed simultaneously.

PS 300 COMMAND AND SYNTAX

```
END;
```

END_S

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PEnds (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used with the PBEGINS procedure to group a set of viewing and/or modeling commands so that each element does not need to be explicitly named to be accessed.

PS 300 COMMAND AND SYNTAX

```
END_Structure;
```

END OPTIMIZE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PEndOpt (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used with the POptStru procedure. When POptStru is called, it places the PS 300 in an "optimization mode" in which certain elements of the display data structure are created in a way that minimizes Display Processor traversal time. PEndOpt must be called to complete the sequence.

It is strongly suggested that users familiarize themselves with the OPTIMIZE command documentation in the **PS 300 Command Summary** before using this procedure to learn the full ramifications and contraints of this command.

PS 300 COMMAND AND SYNTAX

```
END OPTIMIZE;
```

ERASE PATTERN FROM

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PERAPATT ( %DESCR Name1 : P_VaryType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DESCRIPTION

This procedure removes a pattern from name if name is a patterned vector list or curve.

PS 300 COMMAND AND SYNTAX

ERASE PATTERN FROM Name;

Name := EYE BACK

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PEyeBack (   %DESCR Name      : P_VaryingType;
                              DistBack  : REAL;
                              DistHoriz : REAL;
                              DistVert  : REAL;
                              Wide      : REAL;
                              Front     : REAL;
                              Back      : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies a viewing pyramid with the following parametric definitions:

- DistBack is the perpendicular distance of the eye from the plane of the viewport
- DistHoriz is the distance of the eye right or left from the viewport center (positive for right/negative for left)
- DistVert is the distance from the eye up or down from the viewport center (positive for up/negative for down)
- Wide is the width of the viewport
- Front is the front boundary of the frustum of the viewing pyramid
- Back is the back boundary of the frustum of the viewing pyramid

PS 300 COMMAND AND SYNTAX

```
Name := EYE BACK distback
        [left]/[right] disthoriz
        [up]/[down] distvert
        width of the viewport
        Front Boundary = front
        Back Boundary  = back
        (APPLied to Apply);
```

(Continued on next page)

Name := EYE BACK

(continued)

NOTE

PS 300 syntax allows specification of both left and right and up and down in the same command, which results in an accumulation of right/left and up/down. PEYEBACK allows only signed real numbers that if positive specify right and up, and if negative specify left and down.

The following example illustrates this point.

Example:

```
eye_spec:=  eye back .6 left 2.5 right 3 up 2.1 down 6 from screen area 2
            wide front=.0001 back=100 then apply;
```

is equivalent to:

```
eye_spec:=  eye back .6 right .5 down -3.9 from screen area 2 wide
            front=.0001 back=100 then apply;
```

and has the same effect as:

```
PEYEBACK ('EYE_SPEC', 0.6,0.5,-3.9,2,0.0001,100,'APPLY',Error_Handler);
```

Name := F:FUNCTION NAME

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PFnInst (   %DESCR Name      : P_VaryingType;
                      %DESCR FcnName   : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates an instance of an intrinsic PS 300 function.

PS 300 COMMAND AND SYNTAX

Name := F:FcnName;

Name := F:FUNCTION NAME (INOUTS)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PFnInstN (   %DESCR Name      : P_VaryingType;
                       %DESCR FcnName   : P_VaryingType;
                              In_Outs   : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates an instance of an intrinsic PS 300 function where either the number of inputs or outputs is user-defined. The function name (Name) is followed by a number (In_Outs) that describes the number of inputs or outputs to be created with that function. Intrinsic functions that are used by this procedure are F:Route(n), F:RouteC(n), F:Inputs_Choose(n), and F:SYNC(n).

PS 300 COMMAND AND SYNTAX

Name := F:FcnName (InOuts);

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure follows a named operation node (Name) with another operation node. To use the PFOLL procedure, the user must first call this procedure and then the user MUST IMMEDIATELY call the procedure corresponding to the "transformation-or-attribute command".

PS 300 COMMAND AND SYNTAX

FOLLOW name WITH transformation-or-attribute command;

Example:

PS 300 command:

Follow xrot with scale by .5;

would be

```
VAR Vector : P_VectorType;
     .
     .
     .
     .
VECTOR.V4[1]:= 0.5;
VECTOR.V4[2]:= 0.5;
VECTOR.V4[3]:= 0.5;
PFOLL ('xrot',Error-Handler);
PSCALEBY ('',V,'',Error-Handler);
```

Name := CHARACTER FONT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PFont (   %DESCR Name      : P_VaryingType;
                    %DESCR FontName  : P_VaryingType;
                    %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure establishes a character font (FontName) as the working font for the specified display data structure (AppliedTo).

PS 300 COMMAND AND SYNTAX

Name := CHARACTER FONT FontName (APPLied to AppliedTo);

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure removes (Name) from the display and from the name directory, where (Name) is any previously defined data structure name.

PS 300 COMMAND AND SYNTAX

FORget Name;

Name := FIELD_OF_VIEW

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PFov (   %DESCR Name      : P_VaryingType;
                          Angle     : REAL;
                          Front     : REAL;
                          Back      : REAL;
                   %DESCR AppliedTo : P_VaryingType;
                 PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies a right rectangular viewing pyramid with the following parametric definitions:

- Angle is the angle of view from the eye in degrees
- Front is the front boundary of the frustum of the viewing pyramid
- Back is the back boundary of the frustum of the viewing pyramid

PS 300 COMMAND AND SYNTAX

```
Name := Field_Of_View Angle
        FRONT boundary = Front
        BACK boundary  = Back
        (APPLied to AppliedTo);
```

Name := IF CONDITIONAL_BIT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PIfBit (   %DESCR Name       : P_VaryingType;
                            BitNum     : INTEGER;
                            OnOff      : BOOLEAN;
                     %DESCR AppliedTo  : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure refers to a data structure if an attribute bit has a specified setting (On or Off), with the following parametric definitions:

- BitNum indicates which bit to test
- OnOff is TRUE for ON and FALSE for OFF

PS 300 COMMAND AND SYNTAX

Name := IF conditional_BIT BitNum is OnOff (THEN AppliedTo);

Name := IF LEVEL_OF DETAIL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PIfLevel (   %DESCR Name       : P_VaryingType;
                              Level      : INTEGER;
                              Comparison: INTEGER;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure refers to a data structure if the level of detail attribute has a specified relationship to a given number, with the following parametric definitions:

- Level indicates the number to compare with the current level of detail
- *Comparison corresponds to the comparison test to be performed.

PS 300 COMMAND AND SYNTAX

Name := IF LEVEL_of_detail Comp Level (THEN AppliedTo);

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A short table of the mnemonics and their INTEGER value is given below.

Mnemonic	Comparision	INTEGER Value
P_LES	<	0
P_EQL	=	1
P_LEQL	<=	2
P_GTR	>	3
P_NEQL	<>	4
P_GEQL	>=	5

Name := IF PHASE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PIfPhase (   %DESCR Name      : P_VaryingType;
                              OnOff     : BOOLEAN;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure refers to a data structure if the PHASE attribute is in the specified state, ON or OFF. It has the following parametric definition:

- Onoff is TRUE for On and FALSE for Off

PS 300 COMMAND AND SYNTAX

Name := IF PHASE OnOff (THEN AppliedTo);

Name := ILLUMINATION

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PILLUMIN ( %DESCR Name       : P_VaryingType;
                            X          : REAL;
                            Y          : REAL;
                            Z          : REAL;
                            Hue        : REAL;
                            Saturation : REAL;
                            Intensity  : REAL;
                            Ambient    : REAL:   {default 1}
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

DESCRIPTION

This procedure defines polygon illumination characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. The direction to the light source is specified by x, y, z. The color is specified by Hue, Sat and Intens. Its contribution to ambient lighting is specified by Ambien (0 to 1).

PS 300 COMMAND AND SYNTAX

```
Name := ILLUMINATION X, Y, Z
                     [COLOR Hue[,Sat[,Intens]]]
                     [AMBIENT Ambien];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PIncl (   %DESCR Name1      : P_VaryingType;
                    %DESCR Name2      : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used to include one named display data structure (Name1) in a named instance of another display data structure (Name2).

PS 300 COMMAND AND SYNTAX

INCLude Name1 IN Name2;

Name := INCREMENT LEVEL_OF_DETAIL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PIncLOD (   %DESCR Name      : P_VaryingType;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure increments the current level of detail by 1.

PS 300 COMMAND AND SYNTAX

Name := INCRement LEVel_of_detail (APPLied to AppliedTo);

INITIALIZE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PInit (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure restores the PS 300 to its initial state; there are no user-defined names, display data structures, or function connections, and no data structures are displayed.

PS 300 COMMAND AND SYNTAX

INITialize;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PInitC (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure breaks all user-defined function connections.

PS 300 COMMAND AND SYNTAX

INITialize CONNections;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PInitD (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure removes all display data structures from the display list.

PS 300 COMMAND AND SYNTAX

INITialize DISPlays;

INITIALIZE NAMES

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PInitN (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure clears the name dictionary of all display data structure and function instance names.

PS 300 COMMAND AND SYNTAX

INITialize NAMES;

Name1:= INSTANCE OF

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PInst (   %DESCR Name1      : P_VaryingType;
                    %DESCR Name2      : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates an instance node (Name1) with pointers to the data structure referenced (Name2).

PS 300 COMMAND AND SYNTAX

Name1:= INSTance (of Name2);

Name := LABELS (no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PLabAdd (          X          : REAL;
                             Y          : REAL;
                             Z          : REAL;
                   %DESCR Str           : P_VaryingType;
                 PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is the middle call in creating a label block. It must be called to add a label to a previously opened label block created by the call to: PLaBegn. To create a label block, the user must call the procedures:

PLabBegn
PLabAdd (This proeedure may be called multiple times)
PLabEnd

PS 300 COMMAND AND SYNTAX

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

```
Name :=  LABELS x, y, z, 'string'
                  :
                  :
                x, y, z, 'string';
```

Name := LABELS (no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PLabBegn (   %DESCR LabelBlock: P_VaryingType;
                              StepX     : REAL;
                              StepY     : REAL;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to create and open a label block. To complete the label block call and specify a label block, the user must use:

PLabBegn
PLabAdd (This procedure may be called multiple times)
PLabEnd

PS 300 COMMAND AND SYNTAX

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

```
Name := LABELS x, y, z, 'string'
                  :
                  :
                x, y, z, 'string';
```

NOTE

The stepx and stepy parameters allow the steps between the label blocks to be specified in terms of x and y. If stepx and stepy were specified as 1 and 0 respectively, each successive character would be displayed one unit to the right of and horizontally aligned with the preceding character. This applies to all labels within the label block. It should prove useful for those users who wish to make vertical or slanted label blocks. Users cannot send to <step> of a label block; a message from the CI results.

Name := LABELS (no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PLabEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to complete the creation of a label block. To completely specify a label block, the user must call the procedures:

PLabBegn,
PLabAdd (This procedure may be called multiple times), and lastly,
PLabEnd.

PS 300 COMMAND AND SYNTAX

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

```
Name := LABELS x, y, z, 'string'
                  :
                  :
                x, y, z, 'string';
```

Name := LOOK_AT_FROM

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PLookAt (   %DESCR Name      : P_VaryingType;
                         VAR At        : P_VectorType;
                         VAR From      : P_VectorType;
                         VAR Up        : P_VectorType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure, when used with PWINDOW, PEYEBACK, or PFOV, fully specifies the portion of the data space that will be viewed as well as the viewer's orientation in data space. It has the following parametric definitions:

- At is the point being looked at in data space coordinates
- From is the location of the viewer's eye in data space coordinates
- Up indicates the screen "up" direction

PS 300 COMMAND AND SYNTAX

Name := LOOK AT At FROM From UP Up (APPLied to AppliedTo);

Name := MATRIX_2x2

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PMat2x2 (   %DESCR Name      : P_VaryingType;
                        VAR Mat        : P_MatrixType;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a special 2x2 transformation matrix that applies to the specified data (vector list and/or characters) that follow (AppliedTo).

PS 300 COMMAND AND SYNTAX

Name := Matrix_2x2 Mat (APPLied to AppliedTo);

Name := MATRIX_3x3

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PMat3x3 (   %DESCR Name      : P_VaryingType;
                         VAR Mat       : P_MatrixType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a special 3x3 transformation matrix that applies to the specified data (vector lists and/or characters) that follow (AppliedTo).

PS 300 COMMAND AND SYNTAX

Name := Matrix_3x3 Mat (APPLied to AppliedTo);

Name := MATRIX_4x3

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PMat4x3 (   %DESCR Name       : P_VaryingType;
                        VAR Mat         : P_MatrixType;
                        VAR Vec         : P_VectorType;
                      %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a special 4x3 matrix that applies to the specified data (vector lists and/or characters) that follow (AppliedTo).

The Matrix_4x3 command is sent in two parts:

1) a 3x3 matrix is sent in Mat
2) a 3d-translation vector (4th row) is sent in Vec

PS 300 COMMAND AND SYNTAX

Name := Matrix_4x3 Mat Vec (APPLied to AppliedTo);

Name := MATRIX_4x4

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PMat4x4 (   %DESCR Name      : P_VaryingType;
                         VAR Mat       : P_MatrixType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a special 4x4 matrix that applies to the specified data (vector lists and/or characters) that follow (AppliedTo).

PS 300 COMMAND AND SYNTAX

Name := Matrix_4x4 Mat (APPLied to AppliedTo);

Name := NIL

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure names a null data structure. When this procedure is used to redefine (Name), (Name) is kept in the name directory but any definition previously associated with (Name) is removed. PForget does just the opposite of PNameNil.

PS 300 COMMAND AND SYNTAX

Name := NIL;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE POptStru (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used with the PEndOpt procedure. When POptStru is called, it places the PS 300 in an "optimization mode" in which certain elements of the display data structure are created in a way that minimizes Display Processor traversal time. PEndOpt must be called to complete the sequence.

It is strongly suggested that users familiarize themselves with the OPTIMIZE command documentation in the **PS 300 Command Summary** before using this procedure to learn the full ramifications and contraints of this command.

PS 300 COMMAND AND SYNTAX

```
OPTIMIZE STRUCTURE;
```

PATTERN Name1 WITH Name2

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPATWITH ( %DESCR Name1              : P_VaryingType;
                     %DESCR PatternName        : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure patterns the curve of the vector_list called Name with the pattern Patnam, where Patnam has been defined with a call to the procedure PDEFPA.

PS 300 COMMAND AND SYNTAX

PATTERN Name WITH Patnam;

Name := POLYGON (ATTRIBUTES - no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGATR ( %DESCR Attr : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure specifies that the attributes named by Attr and specified in a call to PATTR or PATTR2 apply to all subsequent polygons until superceded by another call to PPLYGA.

This procedure is one of five procedures used to implement the PS_340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :        :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYGON (BEGIN - no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure begins a polygon display list. The parameter (Name) specifies the name to be given to the polygon display list defined by PPLYGA, PPLYGO AND PPLYGL.

This procedure is one of five procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :        :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

A sequence of 3 to 5 procedures must be called to create a polygon display vector list:

PPLYGB: This procedure is called to begin the creation of a polygon vector list.

PPLYGA: This is an optinal procedure called to specify the attribute to be applied to the polygon.

PPLYGO: This is an optional procedure called to specify the intensity or color of the polygon on the calligraphic display.

PPLYGL: This procedure specifies the vectors of each polygon in the polygon display list.

PPLYGE: This procedure closes the polygon display list.

Name := POLYGON (END - no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygEnd  (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure ends the definition of a polygon display list. This procedure is one of five procedures required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYGON (LIST - no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygLis  (       Coplanar  : BOOLEAN;
                            NVertices : INTEGER:
                        VAR Vertices  : P_VectorListType;
                            NormSpec  : BOOLEAN;
                        VAR Normals   : P_VectorListType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar

- NVertices specifies the number of vertices in the polygon

- Vertices specifies the vertices of the polygon
  Vertices [n].Draw = False defines the edge as 'soft'
  Vertices [n].Draw = True defines the edge as 'hard'
  Vertices [n].V4[1] = vertex n: x-coordinate;
  Vertices [n].V4[2] = vertex n: y-coordinate;
  Vertices [n].V4[3] = vertex n: z-coordinate;

- NormSpec specifies if the normals to the vectors defining the polygon are specified. It is TRUE if normals are specified in the Normals array. Otherwise NormSpec = FALSE. At the present time, the runtime software does not support this option. This parameter is presently ignored and reserved for future use.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices. This parameter is reserved for future use when Normals are supported by the runtime software.

Name := POLYGON (LIST - no corresponding command)

(continued)

This procedure is one of five procedures required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYGON (OUTLINE - no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGOTL ( %DESCR Outline : REAL;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DESCRIPTION

This procedure specifies that Outlin be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges on the calligraphic display until superceded by another call to PPLYGO.

This procedure is one of five procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYNOMIAL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPoly (   %DESCR Name      : P_VaryingType;
                           Order     : INTEGER;
                           Dimension : INTEGER;
                       VAR Coeffs    : P_VectorListType;
                           Chords    : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure allows the parametric description of many curve forms without the need to specify or transfer the coordinates of each constituent vector. It has the following parametric definitions:

- Order is the order of the polynomial
- Dimension is either 2 or 3 (2 or 3 dimensions respectively)
- Coeffs represent the x,y,z components of the curve

  where: Coeffs [i].V4 [1]:= x(order -i+1)
  Coeffs [i].V4 [2]:= y(order -i+1)
  Coeffs [i].V4 [3]:= z(order -i+1)
  Coeffs [i].V4 [4] is not used

  To further clarify the description:

  Coeffs [1].V4 [1] := the coefficient that will be applied to the t^{order} term

  Coeffs [2].V4 [1] := the coefficient that will be applied to the $t^{order-1}$ term in the resultant x(t) function computed by this command.

  :
  :
  etc.

- Chords is the number of vectors to be created

Name := POLYNOMIAL

(continued)

PS 300 COMMAND AND SYNTAX

```
Name := POLYNOMIAL
        ORDER = Order
        COEFFICIENTS= X(i),   Y(i),   Z(i)
                      X(i-1), Y(i-1), Z(i-1)
                       :      :       :
                      X(0),   Y(0),   Z(0)
        CHORDS = Chords;
```

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure prefixes a named display structure (Name) with an operation node. To prefix something, the user must first call this procedure and then the user MUST IMMEDIATELY call the procedure corresponding to the: "transformation-or-attribute" command.

PS 300 COMMAND AND SYNTAX

PREfix Name WITH transformation-or-attribute command;

Example:

PS 300 command:

```
Prefix xrot with rotate in z 45;
```

would be

```
PPREF ('xrot', Error-Handler);
PROTZ ('', 45, Error-Handler);
```

SET PIXEL LOCATION - RASTER ROUTINE

RASTER PROCEDURES AND PARAMETERS

```
PROCEDURE PRASCP (      x  : INTEGER;
                        y  : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure establishes the current pixel location relative to the current logical device coordinates. (x) and (y) specify the x,y coordinates of the current pixel and must be greater than or equal to 0.

(0,0) is the lower-left corner of the logical device coordinates.

ERASE_RASTER_SCREEN - RASTER ROUTINE

RASTER PROCEDURE AND PARAMETERS

```
PROCEDURE PRASER (  COLOR  : P_ColorType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used in WRPIX mode to erase the entire screen to the color specified in the parameter (Color), where:

Color . red is the red index
Color . green is the green index
Color . blue is the blue index

The index refers to the color table that contains the actual value used for display.

SET LOGICAL DEVICE COORDINATES - RASTER ROUTINE

RASTER PROCEDURES AND PARAMETERS

```
PROCEDURE PRASLD (   Xmin : INTEGER;
                     Ymin : INTEGER;
                     Xmax : INTEGER;
                     Ymax : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets the logical device coordinates that are used to position the picture in virtual address space. The raster option has a virtual pixel address space from -32768 to 2047 in both x and y. The portion of this space that is actually displayed is from 0 to 639 in x and from 0 to 479 in y. This procedure can be used to reposition an image in screen space without re-calculation and only retransmission of the data.

SET LOOK_UP_TABLE_RANGE - RASTER ROUTINE

RASTER PROCEDURES AND PARAMETERS

```
PROCEDURE PRASLR (    Min  : INTEGER;
                      Max  : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used in WRLUT mode to to set the Look-Up Table range. This procedure set the limits within which the LUT entries can be changed. (Min) and (Max) set the minimum and maximum range of the Look-up tables: they must be greater than or equal to 0 and less than 256.

WRITE LOOK_UP_TABLE ENTRIES - RASTER ROUTINE

RASTER PROCEDURES AND PARAMETERS

```
PROCEDURE PRASLU (      Num  : INTEGER;
                      Index  : INTEGER;
                  VAR Lutval : P_RunColrArrayType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets the current Look-Up Table location and loads the Look-Up Tables. (Num) specifies the number entries in the Lutval parameter. (Index) specifies the location in the Look-Up Table where the entries will start being loaded and (Lutval) is:

Lutval [x]. count is the repetition count
Lutval [x]. red is the red index
Lutval [x]. green is the green index
Lutval [x]. blue is the blue index

If the index is outside of the range set by PRASLR, the values are not changed in this location.

ENABLE/DISABLE RASTER VIDEO - RASTER ROUTINE

RASTER PROCEDURES AND PARAMETERS

```
PROCEDURE PRASVI (    OnOff : BOOLEAN;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used to turn the Raster video on and off. (OnOff) is TRUE to turn the video on, and FALSE to turn the video off.

RASTER PROCEDURES AND PARAMETERS

```
PROCEDURE PRASWP (          Num : INTEGER;
                    VAR Pixval : P_RunClrArrayType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure loads the current pixel location with the pixel values. (Num) specifies the number of entries in (Pixval). (Pixval) is:

Pixval [x]. count is the repetition count
Pixval [x]. red is the red index
Pixval [x]. green is the green index
Pixval [x]. blue is the blue index.

Name := RAWBLOCK

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRAWBLOC ( %DESCR Name          : P_VaryingType;
                            Size          : INTEGER;
                     %DESCR AppliedTo : P_VaryingType
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure creates a structure consisting of a block of contiguous memory with a length of Size bytes.

PS 300 COMMAND AND SYNTAX

Name := RAWBLOCK Size (APPLIED TO Apply);

Name := RATIONAL BSPLINE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRBspl (   %DESCR Name         : P_VaryingType;
                            Order        : INTEGER;
                            OpenClosed   : BOOLEAN;
                            NonPer_Per   : BOOLEAN;
                            Dim          : INTEGER;
                            N_Vertices   : INTEGER;
                            VAR Vertices: P_VectorListType;
                            KnotCount    : INTEGER;
                            VAR Knots    : P_KnotArrayType;
                            Chords       : INTEGER;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure allows the parametric description of a rational B-spline curve form without having to specify or transfer the coordinates of each constituent vector. It contains the following parametric definitions:

- Name specifies the name to be given to the computed rational B-spline
- Order is the order of the curve
- OpenClosed is TRUE for Open and FALSE for Closed
- NonPer_Per is TRUE for Non/periodic and FALSE for Periodic
- Dim is 2 or 3 (2 or 3 dimensions respectively)
- N-Vertices specifies the number of vertices
- Vertices specifies the vertices
- KnotCount is the number of knots
- Knots is the knot sequence
- Chords is the number of vectors to be created

PS 300 COMMAND AND SYNTAX

```
Name :=  RATIONAL BSPLINE
         ORDER = Order
         OPEN/CLOSED
         NONPERIODIC/PERIODIC
         N = NVert
         VERTICES        = X(1), Y(1), ( Z(1), ) W(1)
                           X(2), Y(2), ( Z(2), ) W(2)
                            :    :      :
                           X(N), Y(N), ( Z(N), ) W(N)
         KNOTS = Knots (1), ... Knots (KntCnt)
         CHORDS = Chords;
```

(Continued on next page)

(continued)

NOTE

None of the parameters in the application procedure PRBSPL are optional. The dimension must be specified in the PRBSPL application procedure. In the PS 300 command, dimension is implied by syntax.

If KnotCount = 0, then the default knot sequence is generated and the knot array is ignored.

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure removes (Name) from the display list.

PS 300 COMMAND AND SYNTAX

REMove Name;

REMOVE FOLLOWER of name

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure removes a previously placed 'follower' of (Name).

PS 300 COMMAND AND SYNTAX

REMove FOLLOWER of name;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRemFrom (   %DESCR Name1      : P_VaryingType;
                       %DESCR Name2      : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure removes an instance of a named display data structure (Name1) from an instance node (Name2).

PS 300 COMMAND AND SYNTAX

REMove Name1 FROM Name2;

REMOVE PREFIX

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure removes a previously placed prefix.

PS 300 COMMAND AND SYNTAX

REMove PREfix of name;

Name := ROTATE in X

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRotX (   %DESCR Name      : P_VaryingType;
                           Angle     : REAL;
                    %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a 3x3 rotation matrix that rotates an object (AppliedTo) around the x axis relative to world space origin. It has the following parametric definition:

- Angle is the x rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in X Angle (APPLied to AppliedTo);

Name := ROTATE in Y

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRotY (   %DESCR Name      : P_VaryingType;
                           Angle     : REAL;
                    %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a 3x3 rotation matrix that rotates an object (AppliedTo) around the y axis relative to world space origin. It has the following parametric definition:

- Angle is the y rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in Y Angle (APPLied to AppliedTo);

Name := ROTATE in Z

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRotZ (   %DESCR Name      : P_VaryingType;
                           Angle     : REAL;
                    %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a 3x3 rotation matrix that rotates an object (AppliedTo) around the z axis relative to world space origin. It has the following parametric definition:

- Angle is the z rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in Z Angle (APPLied to AppliedTo);

Name := RATIONAL POLYNOMIAL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRPoly (   %DESCR Name       : P_VaryingType;
                          Order      : INTEGER;
                          Dimension  : INTEGER;
                     VAR  Coeffs     : P_VectorListType;
                          Chords     : INTEGER;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure allows the parametric description of many curve forms without having to specify or transfer the coordinates of each constituent vector. It includes the following parametric definitions:

- Order is the order of the polynomial
- Dimension is 2 or 3 (2 or 3 dimensions respectively)
- Coeffs represent the x,y,z components of the curve

  where: Coeffs [i].V4 [1]:= x(order -i+1)
  Coeffs [i].V4 [2]:= y(order -i+1)
  Coeffs [i].V4 [3]:= z(order -i+1)
  Coeffs [i].V4 [4]:= w(order -i+1)

  To further clarify the description:

  Coeffs [1].V4 [1] := the coefficient that will be applied to the t^{order} term

  Coeffs [2].V4 [1] := the coefficient that will be applied to the $t^{order-1}$ term in the resultant x(t) function computed by this command.

  :
  :
  etc.

- Chords is the number of vectors to be drawn

Name := RATIONAL POLYNOMIAL

(continued)

PS 300 COMMAND AND SYNTAX

```
Name := RATIONAL POLYNOMIAL
        ORDER = Order
        COEFFICIENTS= X(i),   Y(i),   Z(i),   W(i)
                      X(i-1), Y(i-1), Z(i-1), W(i-1)
                        :       :       :       :
                      X(0),   Y(0),   Z(0),   W(0)
        CHORDS = Chords;
```

RESERVE_WORKING_STORAGE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PRsvStor (          Bytes : INTEGER:
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used to reserve working storage space for rendering solids and surfaces. Working storage space must be reserved explicitly using this procedure. The parameter (Bytes) represents the number of bytes to be reserved for working storage.

PS 300 COMMAND AND SYNTAX

Reserve_Working_Storage Bytes;

Name := SCALE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PScaleBy (   %DESCR Name      : P_VaryingType;
                          VAR V         : P_VectorType;
                       %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure applies a scale transformation (V) to a specified vector list and/or characters (AppliedTo). It contains the following parametric definition:

- V is a vector containing the x,y,z scale components

PS 300 COMMAND AND SYNTAX

Name := SCALE by V (APPLied to AppliedTo);

Name := SECTIONING_PLANE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSecPlan  (   %DESCR Name      : P_VaryingType;
                      %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure creates a sectioning-plane node designating that a descendant polygon is a sectioning-plane. The parameter (Name) supplies the name to be given to the sectioning-plane operate node. (AppliedTo) supplies the name of the entity that this node will be applied to.

PS 300 COMMAND AND SYNTAX

Name := SECTIONING_PLANE (Applied to AppliedTo);

Name := SET conditional_BIT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetBit (   %DESCR Name      : P_VaryingType;
                      BitNum    : INTEGER;
                      OnOff     : BOOLEAN;
                    %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure alters one of the 15 global conditional bits during the traversal of the data structure. These conditional bits are initially set to OFF. When the traversal is finished, the bits are restored to their previous values. It contains the following parametric definitions:

- BitNum is an integer from 0 to 14 corresponding to the conditional bit to be set to ON or OFF
- OnOff is TRUE for ON and FALSE for OFF

PS 300 COMMAND AND SYNTAX

Name := SET conditional_BIT BitNum OnOff (APPLied to AppliedTo);

Name := SET COLOR BLENDING

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetBlnd (   %DESCR Name       : P_VaryingType;
                              Saturation: REAL;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure, when used with the ColorBlending parameter of the PVecList procedure, allows individual vector hue saturations to be set. It contains the following parametric definition:

- Saturation is between 0 and 1, where 0 represents no color saturation and 1 represents full color saturation

PS 300 COMMAND AND SYNTAX

Name := SET COLOR BLENDING Saturation (Applied to AppliedTo);

Name := SET CHARACTERS SCREEN_oriented/FIXED

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetChrF (   %DESCR Name      : P_VaryingType;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets the type of screen orientation for displayed character strings. When PSetChrF is used, characters are not affected by rotation or scaling transformations and they are displayed with full size and intensity.

PS 300 COMMAND AND SYNTAX

Name := SET CHARacters SCREEN_oriented/FIXED (APPLied to AppliedTo);

Name := SET CHARACTERS SCREEN_oriented

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetChrS (   %DESCR Name      : P_VaryingType;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets the type of screen orientation for displayed character strings. When PSetChrS is used, characters are not affected by rotation or scaling transformations, but intensity and size will still vary with depth (Z-position).

PS 300 COMMAND AND SYNTAX

Name := SET CHARacters SCREEN_oriented (APPLied to AppliedTo);

Name : SET CHARACTERS WORLD_ORIENTED

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetChrW (   %DESCR Name      : P_VaryingType;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets the type of screen orientation for displayed character strings. When PSetChrW is used, characters are transformed along with any part of the object containing them.

PS 300 COMMAND AND SYNTAX

Name := SET CHARacters WORLD_oriented (APPLied to AppliedTo);

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSETCNES (      Cness      : BOOLEAN;
                          Inp        : INTEGER;
                    %DESCR Name : P_VaryingType;
                   PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedue is used to define a particular function instance input to be a constant or trigger input.

PS 300 COMMAND AND SYNTAX

SETUP CNESS TRUE <Inp> Name;
SETUP CNESS FALSE <Inp> Name;

Name := SET COLOR

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetColr (   %DESCR Name       : P_VaryingType;
                              Hue        : REAL;
                              Saturation: REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the color of an object (AppliedTo). It contains the following parametric definitions:

- Hue is greater than or equal to 0 and less than 360 with:
  0 = pure blue
  120 = pure red
  240 = pure green

- Saturation is from 0 to 1 with:
  0 = no saturation (white)
  1 = full saturation

PS 300 COMMAND AND SYNTAX

Name := SET COLOR Hue,Sat (APPLied to AppliedTo);

Name := SET CONTRAST

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetCont (   %DESCR Name      : P_VaryingType;
                             Contrast  : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure changes the contrast of the display data structure (AppliedTo). It contains the following parametric definition:

- Contrast is between 0 and 1, where 0 represents the lowest contrast and 1 represents the highest contrast

PS 300 COMMAND AND SYNTAX

Name := SET CONTrast to Contrast (APPLied to AppliedTo);

Name := SET CSM

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetCSM (   %DESCR Name       : P_VaryingType;
                             OnOff      : BOOLEAN;
                      %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure allows the CSM to be set to ON or OFF; ON provides extra brightness and precision, OFF is the default setting and allows for the maximum number of vectors to be displayed. It contains the following parametric definition:

- OnOff is TRUE for On and FALSE for Off

PS 300 COMMAND AND SYNTAX

Name := SET CSM OnOff (APPLied to AppliedTo);

Name := SET DISPLAYS ALL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetDAll (   %DESCR Name       : P_VaryingType;
                              OnOff      : BOOLEAN;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets all display(s) to ON or OFF. It has the following parametric definition:

- OnOff is TRUE for On and FALSE for Off

PS 300 COMMAND AND SYNTAX

Name := SET DISPlays ALL OnOff (APPLied to AppliedTo);

Name := SET DEPTH_CLIPPING

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetDCL (   %DESCR Name       : P_VaryingType;
                             OnOff      : BOOLEAN;
                      %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure enables/disables depth clipping. With depth clipping Off, data between the front clipping plane and the eye will appear at full intensity and data behind the eye will be clipped. It has the following parametric definition:

- OnOff is TRUE for On and FALSE for Off

PS 300 COMMAND AND SYNTAX

Name := SET DEPTH_CLipping OnOff (APPLied to AppliedTo);

Name := SET DISPLAY

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetDOnF (   %DESCR Name       : P_VaryingType;
                              OnOff      : BOOLEAN;
                              N          : INTEGER;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the display to be set to On or Off and has the following parametric definitions:

- N is the number of the display to be set to On or Off
- OnOff is TRUE for On and FALSE for Off

PS 300 COMMAND AND SYNTAX

Name := SET DISPlay N OnOff (APPLied to AppliedTo);

Name := SET INTENSITY

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetInt (   %DESCR Name       : P_VaryingType;
                             OnOff      : BOOLEAN;
                             Imin       : REAL;
                             Imax       : REAL;
                      %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the intensity variation for depth cueing and has the following parametric definitions:

- OnOff is TRUE for On and FALSE for Off
- IMin is real number from 0.0 to 1.0 that represents the dimmest intensity setting
- IMax is a real number from 0.0 to 1.0 that represents the brightest intensity setting.

PS 300 COMMAND AND SYNTAX

Name := SET INTENSITY OnOff IMin:IMax (APPLied to AppliedTo);

Name := SET LEVEL_OF_DETAIL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetLOD (   %DESCR Name      : P_VaryingType;
                             Level     : INTEGER;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure alters a global level of detail value temporarily. These temporary settings allow for conditional referencing to other data structures. When the traversal of data is finished, the level of detail is restored to its original level. It contains the following parametric definition:

- Level is an integer from 0 to 32767 that indicates the level of detail value

PS 300 COMMAND AND SYNTAX

Name := SET LEVel_of_detail TO Level (APPLied to AppliedTo);

Name := SET PICKING IDENTIFIER

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetPID (   %DESCR Name      : P_VaryingType;
                      %DESCR PickId    : P_VaryingType;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies textual information that will be reported back if a pick occurs on the specified display data structure (AppliedTo). It contains the following parametric definition:

- PickId is the text that will be reported if a pick occurs anywhere within the structure (AppliedTo)

PS 300 COMMAND AND SYNTAX

Name := SET PICKing IDentifier = PickId (APPLied to AppliedTo);

Name := SET PICKING LOCATION

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetPLoc (   %DESCR Name      : P_VaryingType;
                              Xcenter   : REAL;
                              Ycenter   : REAL;
                              Xsize     : REAL;
                              Ysize     : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies a rectangular picking area at (x,y) within the current viewport. It contains the following parametric definitions:

- XCenter, YCenter signify the center of the pick location
- Xsize, Ysize specify the boundaries of the pick rectangle

PS 300 COMMAND AND SYNTAX

Name := SET PICKING LOCation = XCenter, YCenter, Xsize, Ysize (APPLied to AppliedTo);

Name := SET PLOTTER

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSETPLOT ( %DESCR Name      : P_VaryingType;
                            OnOff     : BOOLEAN;
                     %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure enables or disables the plotting of subsequent nodes in the data structure.

PS 300 COMMAND AND SYNTAX

Name := SET PLOTTER OnOff (APPLIED TO Apply);

Name := SET PICKING Switch

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetPOnf (   %DESCR Name      : P_VaryingType;
                              OnOff     : BOOLEAN;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure enables/disables picking for a specified display data structure (AppliedTo). It contains the following parametric definition:

- OnOff is TRUE for On and FALSE for Off

PS 300 COMMAND AND SYNTAX

Name := SET PICKing OnOff (APPLied to AppliedTo);

Name := SET RATE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetR (   %DESCR Name      : P_VaryingType;
                           PhaseOn   : INTEGER;
                           PhaseOff  : INTEGER;
                           InitOnOff : BOOLEAN;
                           Delay     : INTEGER;
                    %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets two global duration values (PhaseOn and PhaseOff) during the traversal of a specified data structure (AppliedTo). The default phase is off and never changes unless a SET RATE node is encountered. The procedure has the following parametric definitions:

- PhaseOn designates the duration of the ON phase
- PhaseOff designates the duration of the OFF phase
- InitOnOff is TRUE for On and FALSE for Off
- Delay is the number of refresh frames in the initial state

PS 300 COMMAND AND SYNTAX

Name := SET RATE PhaOn PhaOff IniOnF Delay (APPLied to AppliedTo);

Name := SET RATE EXTERNAL

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetRExt (    %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sets up a data structure that can be used to alter the Phase attribute using an external source, such as a function network or a message from the host computer.

PS 300 COMMAND AND SYNTAX

Name := SET RATE EXTernal (APPlied to AppliedTo);

SEND BOOLEAN TO

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndBool (          B          : BOOLEAN;
                              Inp        : INTEGER;
                    %DESCR    Dest       : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a Boolean value to input (Inp) of a specified function instance, display data structure, or variable (Dest). It has the following parametric definitions:

- B is the Boolean value to be sent, TRUE or FALSE
- *Inp is the input of the display data structure, function instance, or variable

PS 300 COMMAND AND SYNTAX

SEND TRUE TO ‹Inp› Dest;
SEND FALSE TO ‹Inp› Dest;

* This mnemonic may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_LAST	‹LAST›	-5

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndFix (          i          : INTEGER;
                             Inp        : INTEGER;
                    %DESCR   Dest       : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends the value of (i) to the specified input (Inp) of the display data structure, function instance, or variable (Dest). It has the following parametric definitions:

- i is the integer to be sent
- *Inp is an INTEGER corresponding to the input of a display data structure, function instance, or variable

PS 300 COMMAND AND SYNTAX

SEND FIX (i) TO ‹Inp› Dest;

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_Delete	‹DELETE›	-1
P_Clear	‹CLEAR›	-2

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndM2d (      VAR Mat        : P_MatrixType;
                             Inp        : INTEGER;
                        %DESCR Dest     : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a 2x2 matrix to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND M2D (Mat) TO <Inp> Dest;

SEND 3D MATRIX TO

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndM3d (      VAR Mat        : P_MatrixType;
                             Inp        : INTEGER;
                        %DESCR Dest     : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a 3x3 matrix to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND M3D (Mat) TO ‹Inp› Dest;

SEND 4D MATRIX TO

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndM4d (      VAR Mat        : P_MatrixType;
                             Inp        : INTEGER;
                        %DESCR Dest     : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a 4x4 matrix to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND M4D (Mat) TO ‹Inp› Dest;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndPL (          Count      : INTEGER;
                            DrawMove   : BOOLEAN;
                            Inp        : INTEGER;
                       %DESCR Name     : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies (Count) number of Draws or Moves to be sent to input (Inp) of a vector list (Dest). It contains the following parametric definitions:

- Count is the number of Draws/Moves
- DrawMove is TRUE for Draw and FALSE for Move

PS 300 COMMAND AND SYNTAX

SEND Count*DrawMove TO ‹Inp› Dest;

SEND Real-number TO

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndReal (          r          : REAL;
                              inp        : INTEGER;
                    %DESCR Dest          : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a real number (r) to a specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND r TO ‹Inp› Dest;

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndStr (   %DESCR Str        : P_VaryingType;
                             Inp        : INTEGER;
                      %DESCR Dest       : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends the character string (Str) to input (Inp) of (Dest).

PS 300 COMMAND AND SYNTAX

SEND 'Str' TO ‹Inp› Dest;

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_LAST	‹LAST›	-5
P_Substitute	‹SUBSTITUTE›	-6

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndV2d (      VAR V          : P_VectorType;
                             Inp        : INTEGER;
                     %DESCR Dest        : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a 2D vector to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND V2D (V) TO ‹Inp› Dest;

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_Append	‹APPEND›	0
P_Step	‹STEP›	-3
P_Position	‹POSITION›	-4
P_LAST	‹LAST›	-5

SEND 3D VECTOR TO

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndV3d (      VAR V           : P_VectorType;
                             Inp         : INTEGER;
                        %DESCR Dest      : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a 3D vector to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND V3D (V) TO ‹Inp› Dest;

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_Append	‹APPEND›	0
P_Step	‹STEP›	-3
P_Position	‹POSITION›	-4
P_Last	‹LAST›	-5

SEND 4D VECTOR TO

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndV4d (      VAR V          : P_VectorType;
                             Inp        : INTEGER;
                        %DESCR Dest     : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends a 4D vector to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND V4D (V) TO ‹Inp› Dest;

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_Append	‹APPEND›	0
P_Step	‹STEP›	-3
P_Position	‹POSITION›	-4
P_Last	‹LAST›	-5

SEND VALUE TO (Variable)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndVal (   %DESCR Varname   : P_VaryingType;
                             Inp       : INTEGER;
                      %DESCR Dest      : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure sends the current value in variable (Varname) to input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND VALUE (VarName) TO <Inp> Dest;

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	<Input>	INTEGER Value
P_Append	<APPEND>	0
P_Delete	<DELETE>	-1
P_Clear	<CLEAR>	-2
P_Step	<STEP>	-3
P_Position	<POSITION>	-4
P_Last	<LAST>	-5
P_Substitute	<SUBSTITUTE>	-6

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSndVL (   %DESCR Name1      : P_VaryingType;
                           Inp         : INTEGER;
                     %DESCR Name2      : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure replaces vectors beginning at vector (Inp) of the vector list (Name2) with the vectors from the vector list (Name1).

PS 300 COMMAND AND SYNTAX

SEND VL (Name1) TO ‹Inp› Name2;

* This mnemonic may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure. See the section on Programming Suggestions for a description of PROCONST.PAS. A description of inputs to display data structures and their INTEGER value is given below.

Mnemonic	‹Input›	INTEGER Value
P_Append	‹APPEND›	0
P_Last	‹LAST›	-5

Name := SOLID_RENDERING

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSolRend  (   %DESCR Name      : P_VaryingType;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines a solid-rendering operate node, marking its descendent structure so that solid renderings can be performed on it. The parameter (Name) supplies the name to be given to the solid-rendering operate node. (AppliedTo) supplies the name of the entity that this operate node will be applied to.

PS 300 COMMAND AND SYNTAX

Name := SOLID_RENDERING (Applied to AppliedTo);

Name := SURFACE_RENDERING

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSurRend   (   %DESCR Name      : P_VaryingType;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines a surface-rendering operate node, marking its descendent structure so that surface renderings can be performed on it. The parameter (Name) supplies the name to be given to the surface-rendering operate node. (AppliedTo) supplies the name of the entity that this operate node will be applied to.

PS 300 COMMAND AND SYNTAX

Name := SURFACE_RENDERING (Applied to AppliedTo);

Name := STANDARD FONT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PStdFont (    %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure establishes the standard PS 300 character font as the working font.

PS 300 COMMAND AND SYNTAX

Name := STANdard FONT (APPLied to AppliedTo);

Name := TRANSLATE

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PTransBy (   %DESCR Name      : P_VaryingType;
                          VAR Vec       : P_VectorType;
                       %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure applies a translation vector (Vec) to the specified data structure (AppliedTo).

All three components (x,y,z) must be specified.

Specifically:

Vec.V4 [1]:= X translation;
Vec.V4 [2]:= Y translation;
Vec.V4 [3]:= Z translation;

PS 300 COMMAND AND SYNTAX

Name := TRANslate by Vec (APPLied to AppliedTo);

VARIABLE Name

APPLICATION PROCEDURE AND PARAMETERS

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

DEFINITION

This procedure defines a PS 300 variable where (Name) contains the name of the variable to be created.

PS 300 COMMAND AND SYNTAX

VARiable Name;

Name := VECTOR_LIST (no corresponding command)

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)

Name := VECTOR_LIST (no corresponding command)

(continued)

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

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE) 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

Name := VECTOR_LIST (no corresponding command)

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

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

PVecBegn

PVecList (This procedure 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) N=n <vectors>;

Name := VECTOR_LIST (no corresponding command)

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 many times to send the vector list down in pieces. For block-normalized vector lists, this procedure can only be called once. Multiple calls to this procedure are not permitted for the Block-normalized vector list case. 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) 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
0 <= vectors [n].V4[4] <=1

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 4th position in VECTORS is always the intensity regardless of the dimension of the vector list. In block normalized, the 1st vector's 4th position is the intensity for the entire vector list.

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

Name := VIEWPORT

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PViewP (   %DESCR Name      : P_VaryingType;
                            Xmin      : REAL;
                            Xmax      : REAL;
                            Ymin      : REAL;
                            Ymax      : REAL;
                            Imin      : REAL;
                            Imax      : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It contains the following parametric definitions:

- XMin, Xmax (horizontal) specify the horizontal boundaries of the new viewport
- YMin, Ymax (vertical) specify the vertical boundaries of the new viewport
- IMin, IMax specify the minimum and maximum intensities for the viewport

PS 300 COMMAND AND SYNTAX

```
Name := VIEWport HORizontal = Xmin:Xmax
                 VERTical   = Ymin:Ymax
                 INTENsity  = Imin:Imax
                 (APPLied to AppliedTo);
```

Name := WINDOW

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PWindow (   %DESCR Name      : P_VaryingType;
                             Xmin      : REAL;
                             Xmax      : REAL;
                             Ymin      : REAL;
                             Ymax      : REAL;
                             Front     : REAL;
                             Back      : REAL;
                        %DESCR AppliedTo : P_VaryingType;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies a right rectangular prism enclosing a portion of the data space to be displayed in parallel projection. It contains the following parametric definitions:

- XMin, Xmax (horizontal) specify the window's boundaries along the x axis
- YMin, Ymax (vertical) specify the window's boundaries on the y axis
- Front specifies the front boundary
- Back specifies the back boundary

PS 300 COMMAND AND SYNTAX

```
Name := WINDOW  X = Xmin:Xmax
                Y = Ymin:Ymax
                FRONT boundary = Front
                BACK  boundary = Back
                (APPLied to AppliedTo);
```

Name := CANCEL XFORM

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PXFCANCL ( %DESCR Name       : P_VaryingType;
                     %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure stops transform data processing of subsequent nodes.

PS 300 COMMAND AND SYNTAX

Name := CANCEL XFORM (APPLIED TO Apply);

Name := XFORM MATRIX

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PXFMATRX ( %DESCR Name       : P_VaryingType;
                     %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure allows subsequent nodes to be processed to produce a transformation matrix.

PS 300 COMMAND AND SYNTAX

Name := XFORM MATRIX (APPLIED TO Apply);

Name := XFORM VECTOR_LIST

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PXFVECTR ( %DESCR Name       : P_VaryingType;
                     %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure allows subsequent nodes to be processed to produce a transformation vector_list.

PS 300 COMMAND AND SYNTAX

Name := XFORM VECTOR_LIST (APPLIED TO Apply);

PS 340 GSR PASCAL ERROR DEFINITIONS

The tables listed in this section define the possible error codes used to identify warning or error conditions that may arise while using the Graphics Support Routines.

The set of possible error codes is divided into several regions reserved for specific severity and machine dependency levels:

1...255	= Machine INDEPENDENT warning conditions.
256...511	= Machine DEPENDENT warning conditions.
512...767	= Machine INDEPENDENT error conditions.
768...1023	= Machine DEPENDENT error conditions.
1024...1279	= Machine INDEPENDENT fatal error conditions.
1280...1535	= Machine DEPENDENT fatal error conditions.

ERROR TABLE - 1

The following warning codes allow successful completion of the GSR procedure, but indicate a probable user error.

Error Code	Mnemonic	Meaning
1	PSW_BadNamChr	Bad name character. Any PS 300 invalid name character is translated to the underscore character.
2	PSW_NamTooLon	Name too long. Name truncated to 256 characters.
3	PSW_StrTooLon	String too long. String truncated to 240 characters.
30	PSW_PixCouGre	The Pixel Count is greater than the screen size in call to PRASWP. (Reserved for P6.V01 Raster procedures.)
31	PSW_PixCouLes	The Pixel Count is less than 1 in call to PRASWP. (Reserved for P6.V01 Raster procedures.)
32	PSW_RepCouGre	Repetition count greater than 255 in call to PRASLU. (Reserved for P6.V01 Raster procedures.)
33	PSW_RepCouLes	Repetition count less than 1 in call to PRASLU. (Reserved for P6.V01 Raster procedures.)

Error Code	Mnemonic	Meaning
256	PSW_AttAlrDon	Attach already done.
257	PSW_AtnKeySee	Attention key seen. This tells the error-handling routine that the user hit the Attention key (IBM version only).
258	PSW_BadGenChr	The string specified to be sent to the "generic" output channel of CIROUTE via the PPutGX subroutine contained an invalid character that has been translated to a blank space character. This error code CANNOT be caused by invoking the subroutine: PPutG which does NOT perform any translation on the specified string (IBM version only).
259	PSW_BadStrChr	Bad string character. Any invalid string character is converted to a blank space character.
260	PSW_BadParChr	The string specified to be sent to the PS 300 Parser via the PPutP subroutine contained an invalid character that has been translated to a blank space character.

ERROR TABLE - 2

For the following errors, the GSRs abort the current command sequence (if there is one) and ignore the out-of-sequence command that (probably) caused this error.

Error Code	Mnemonic	Meaning
515	PSE_PreOpeExp	Prefix operate node call expected.
516	PSE_FolOpeExp	Follow operate node call expected.
517	PSE_LabBlkExp	Label block call expected.
518	PSE_VecLisExp	Vector List call expected.
519	PSE_AttMulVec	Attempted multiple PVecList call sequence for block normalized vectors prohibited.
520	PSE_MisLabBeg	Missing Begin Label block call.

Error Code	Mnemonic	Meaning
521	PSE_MisVecBeg	Missing Begin Vector List call.
529	PSE_MisPolBeg	The Begin polygon call, PPlygBeg is missing. PPlygAtr, PPlygLis, or PPlygEnd was called without the prerequisite call to PPLygBeg.
530	PSE_PAtPliPen	A call to PPlygAtr, PPlygLis, or PPlygEnd was expected.
531	PSE_PLiPEnExp	A call to PPlygLis or PPlygEnd was expected.
532	PSE_PAtPLiExp	A call to PPlygAtr or PPlygLis was expected.
533	PSE_PLiExp	A call to PPlygLis was expected.

ERROR TABLE - 3

The following errors are user errors and are generated by invalid parameters or by an unsuccessful attempt to attach.

Error Code	Mnemonic	Meaning
512	PSE_InvMuxCha	Invalid multiplexing channel argument specified in a call to PMuxP, PMuxCI, or PMuxG. The multiplexing channel assigned to the Parser, CI, or Generic channel is not changed.
513	PSE_InvVecCla	Invalid vector list class specified in call to PVecBegn. Command is ignored.
514	PSE_InvVecDim	Invalid vector list dimension specified in call to PVecBegn. Command is ignored.
522	PSE_NulNam	A null name is not permitted in this procedure context. The command is ignored.
523	PSE_BadComTyp	Bad Comparison type operator specified. If Level = command ignored.

Error Code	Mnemonic	Meaning
524	PSE_InvFunNam	Attempted PS 300 function instance call failed because the named function cannot possibly exist. The function name identifying the function type to instance was longer than 256 characters.
525	PSE_NulNamReq	Null name was required for parameter in operate node call following a PPref or PFoll subroutine.
526	PSE_TooManEnd	Too many PEnds calls for the number of preceding PBegs calls. Command ignored.
527	PSE_NotAtt	The PS 300 communications link has not been established. The user failed to call PAttach or an error occurred in the attach procedure preventing the communications link from being created.
528	PSE_OveDurRea	An overrun occurred during a READ operation. The user-supplied input buffer was too small and truncation has occurred.
534	PSE_MaxPolExc	The polygon specified by the call to PPlygLis contains more than 250 vertices. The polygon is ignored.
535	PSE_LesMinPol	The polygon specified by the call to PPLygLis contains fewer than 3 vertices. It is therefore a degenerate polygon and is ignored.
536	PSE_IllPolAtr	Illegal polygon attribute(s) specified in the call to PPlygAtr. Attribute(s) ignored.
550	PSE_IllCurPix	Illegal Current Pixel specification in call to PRASCP. (Reserved for P6.V01 Raster procedures.)
552	PSE_IndOutRan	Index out of range: 0...255 in call to PRASLU. (Reserved for P6.V01 Raster procedures.)
553	PSE_IllLDCpe	Illegal LDC specification in call to PRASLD. (Reserved for P6.V01 Raster procedures.)
554	PSE_SLUNumLes	NUM parameter less than 1 in call to PRASLD. (Reserved for P6.V01 Raster procedures.)

Error Code	Mnemonic	Meaning
555	PSE_MinGreMax	Minimum > Maximum in call to PRASLR. (Reserved for P6.V01 Raster procedures.)
556	PSE_MinOutRan	Minimum out of range 0...255 in call to PRASLR. (Reserved for P6.V01 Raster procedures.)
557	PSE_MaxOutRan	Maximum out of range 0...255 in call to PRASLR. (Reserved for P6.V01 Raster procedures.)
558	PSE_SWPNumLes	NUM parameter less than 1 in call to PRASWP. (Reserved for P6.V01 Raster procedures.)

At the present time, the following three error messages (780, 781, 782) are only meaningful for Digital Equipment Corporation (DEC) VAX/VMS*. All three errors indicate that the parameter passed as a string in PAttch was not successfully parsed and that the Attach call failed.

Error Code	Mnemonic	Meaning
780	PSE_PhyDevTyp	This error indicates that a missing or invalid Physical Device Type was specified in a call to PAttch.
781	PSE_LogDevNam	This error indicates that a missing or invalid Logical Device Name was specified in a call to PAttach.
782	PSE_AttDelExp	This error indicates that an Attach parameter string was expected in a call to PAttach.

FATAL ERRORS

The errors listed below indicate a very serious error condition. If the user's error handler is invoked with any of the error codes listed below, then program execution should be aborted.

*Trademark of the Digital Equipment Corporation, Maynard, Massachusetts

ERROR TABLE - 4

Error Code	Mnemonic	Meaning
1024	PSF_IllFraCom	Illegal frame command specified in call to: PSUTIL_RasMode. This error code indicates an internal validity check error. E&S Software Support should be contacted.
1280	PSF_PhyAttFai	Physical Attach operation failed.
1281	PSF_PhyDetFai	Physical Detach operation failed.
1282	PSF_PhyGetFai	Physical Get operation failed.
1283	PSF_PhyPutFai	Physical Put operation failed.

The following three errors are only applicable to the DEC VAX/VMS version of the GSRs. All three error codes indicate an internal GSR validity error. E&S Software Support should be contacted if these errors are detected.

Error Code	Mnemonic	Meaning
1290	PSF_BufTooLar	Buffer too large in a call to PSPUT. Internal validity check error.
1291	PSF_WroNumArg	Wrong number of arguments to low-level I/O procedure in PROIOLIB.MAR. Internal validity check error.
1292	PSF_ProTooLar	Prompt too large in call to PSFRCV. Internal validity check error.

APPENDIX A. SAMPLE PROGRAMS

This appendix contains sample Pascal programs that illustrate the use of the PS 300 DEC/VAX PASCAL V2 Graphics Support Routines. The programs contain examples of error-handler procedures.

This is a PASCAL network creation example.

```
PROGRAM BlkLevp (INPUT, OUTPUT);

CONST
 Deg_rad = 0.017453292;
 %INCLUDE 'PROCONST.PAS'

TYPE
 %INCLUDE 'PROTYPES.PAS'

VAR
 Front       : P_VectorListType;
 Vecs        : P_VectorListType;
 Zero_vec    : P_VectorType;
 Y_Up        : P_VectorType;
 At          : P_VectorType;
 From        : P_VectorType;
 Up          : P_VectorType;
 Name        : P_VaryingType;
 Theta       : REAL;
 DTheta      : REAL;
 i           : INTEGER;
 k           : INTEGER;
 l           : INTEGER;
 Times       : INTEGER;
```

```
%INCLUDE 'PROEXTRN.PAS'


{ The following Error Handler demonstrates the                }
{ general overall recommended form that the user's            }
{ own error handler should follow.                            }
{                                                             }
{ This error handler upon being invoked writes ALL            }
{ messages to the data file:  'PROERROR.LOG' for 2            }
{ reasons:                                                    }
{                                                             }
{    1.  The error handler should NOT immediately             }
{        write information out on the PS 300 screen           }
{        since the explanatory text defining the              }
{        error or warning condition may be taken as           }
{        data by the PS 300 and therefore wind up             }
{        not being displayed on the PS 300 screen             }
{        (as in the case of a catastrophic data               }
{        transmission error.                                  }
{                                                             }
{    2.  The logging of errors and warnings to a              }
{        logfile allows any errors and/or warnings            }
{        to be reviewed at a later time.                      }


PROCEDURE Err ( Error_code: Integer );

VAR
  VMSdef, PIdef : P_VaryingType;
  Error_Log     : [STATIC] TEXT;
  ErrorFileOpen : [STATIC] BOOLEAN := FALSE;


  [EXTERNAL] PROCEDURE LIB$STOP
          (%IMMED CompletionCode : INTEGER); EXTERN;


  PROCEDURE IBM_Specific;
  BEGIN
    WRITE   (Error_Log, 'This error/warning is ');
    WRITE   (Error_Log, 'applicable ONLY to the IBM ');
    WRITELN (Error_Log, 'version of the');
    WRITELN (Error_Log, 'Procedural Interface (GSR).');
  END;
```

```
PROCEDURE VAX_Specific;
 BEGIN
  WRITE  (Error_Log, 'This error/warning is ');
  WRITE  (Error_Log, 'applicable ONLY to the DEC ');
  WRITELN (Error_Log, 'VAX/VMS version of');
  WRITE  (Error_Log, 'the Procedural Interface ');
  WRITELN (Error_Log, '(GSR).');
 END;


 PROCEDURE UnknownError;
 BEGIN
  WRITE  (Error_Log, 'PS-W-UNRCOMCOD:  ');
  WRITE  (Error_Log, 'Procedural Interface ');
  WRITE  (Error_Log, '(GSR) completion ');
  IF Error_code < 512
    THEN WRITE (Error_Log, 'warning ')
    ELSE IF Error_code < 1024
         THEN WRITE (Error_Log, 'error ')
         ELSE WRITE (Error_Log, 'fatal error ');
  WRITELN (Error_Log, 'code is unrecognized.');
  WRITE  (Error_Log, 'Probable Procedural ');
  WRITE  (Error_Log, 'Interface (GSR) Internal ');
  WRITELN (Error_Log, 'validity check error.');
 END;


 PROCEDURE IdentifyCompletionCode
       (Error_code : INTEGER);
 BEGIN
  WRITE  (Error_Log, 'PS-I-PROERRWAR:  Procedural ');
  WRITE  (Error_Log, 'Interface (GSR) warning/');
  WRITE  (Error_Log, 'error completion code was ');
  WRITELN (Error_Log, 'received.');

  { Identify warning codes }

  IF Error_Code < 512 THEN CASE Error_Code OF
    PSW_BadNamChr:
    BEGIN
      WRITE  (Error_Log, 'PS-W-BADNAMCHR:  Bad ');
      WRITE  (Error_Log, 'character in name was ');
      WRITELN (Error_Log, 'translated to:  "_".');
    END;
    PSW_NamTooLon:
    BEGIN
      WRITE  (Error_Log, 'PS-W-NAMTOOLON:  Name too ');
      WRITE  (Error_Log, 'long.  Name was truncated to ');
```

```
    WRITELN (Error_Log, '256 characters.');
  END;
  PSW_StrTooLon:
  BEGIN
    WRITE  (Error_Log, 'PS-W-STRTOOLON:  String too ');
    WRITE  (Error_Log, 'long.  String was truncated ');
    WRITELN (Error_Log, 'to 240 characters.');
  END;
  PSW_AttAlrDon:
  BEGIN
    WRITE  (Error_Log, 'PS-W-ATTALRDON:  Attach ');
    WRITE  (Error_Log, 'already done.  Multiple call ');
    WRITELN (Error_Log, 'to PAttach without');
    WRITE  (Error_Log, 'intervening PDetach call ');
    WRITELN (Error_Log, 'ignored.');
  END;
  PSW_AtnKeySee:
  BEGIN
    WRITE  (Error_Log, 'PS-W-ATNKEYSEE:  Attention ');
    WRITELN (Error_Log, 'key seen (depressed).');
    IBM_Specific;
  END;
  PSW_BadGenChr:
  BEGIN
    WRITE  (Error_Log, 'PS-W-BADGENCHR:  Bad generic ');
    WRITE  (Error_Log, 'channel character.  Bad ');
    WRITELN (Error_Log, 'character in string sent via:');
    WRITE  (Error_Log, '  PPutGX  was translated to ');
    WRITELN (Error_Log, 'a blank.');
    IBM_Specific;
  END;
  PSW_BadStrChr:
  BEGIN
    WRITE  (Error_Log, 'PS-W-BADSTRCHR:  Bad ');
    WRITE  (Error_Log, 'character in string was ');
    WRITELN (Error_Log, 'translated to a blank.');
    IBM_Specific;
  END;
  PSW_BadParChr:
  BEGIN
    WRITE  (Error_Log, 'PS-W-BADPARCHR:  Bad parser ');
    WRITE  (Error_Log, 'channel character.  Bad ');
    WRITELN (Error_Log, 'character in string sent to');
    WRITE  (Error_Log, 'PS 300 parser via:  PPutP  ');
    WRITELN (Error_Log, 'was translated to a blank.');
    IBM_Specific;
  END;
  OTHERWISE UnknownError;
END
```

```
{ Identify errors }

ELSE IF Error_code < 1024 THEN CASE Error_Code OF
  PSE_InvMuxCha:
  BEGIN
    WRITE   (Error_Log, 'PS-E-INVMUXCHA:  Invalid ');
    WRITE   (Error_Log, 'multiplexing channel ');
    WRITELN (Error_Log, 'specified in call to:');
    WRITELN (Error_Log, 'PMuxCI, PMuxP, or PMuxG.');
  END;
  PSE_InvVecCla:
  BEGIN
    WRITE   (Error_Log, 'PS-E-INVVECCLA:  Invalid ');
    WRITE   (Error_Log, 'vector list class specified ');
    WRITELN (Error_Log, 'in call to:  PVecBegn.');
  END;
  PSE_InvVecDim:
  BEGIN
    WRITE   (Error_Log, 'PS-E-INVVECDIM:  Invalid ');
    WRITE   (Error_Log, 'vector list dimension ');
    WRITELN (Error_Log, 'specified in call to');
    WRITELN (Error_Log, 'PVecBegn.');
  END;
  PSE_PreOpeExp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-PREOPEEXP:  Prefix ');
    WRITELN (Error_Log, 'operator call was expected.');
  END;
  PSE_FolOpeExp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-FOLOPEEXP:  Follow ');
    WRITELN (Error_Log, 'operator call was expected.');
  END;
  PSE_LabBlkExp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-LABBLKEXP:  Call to ');
    WRITE   (Error_Log, 'PLabAdd or PLabEnd was ');
    WRITELN (Error_Log, 'expected.');
  END;
  PSE_VecLisExp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-VECLISEXP:  Call to ');
    WRITE   (Error_Log, 'PVecList or PVecEnd was ');
    WRITELN (Error_Log, 'expected.');
  END;
  PSE_AttMulVec:
  BEGIN
    WRITE   (Error_Log, 'PS-E-ATTMULVEC:  Attempted ');
```

```
  WRITE  (Error_Log, 'multiple call sequence to ');
  WRITELN (Error_Log, 'PVecList is NOT permitted');
  WRITELN (Error_Log, 'for BLOCK normalized vectors.');
END;
PSE_MisLabBeg:
BEGIN
  WRITE  (Error_Log, 'PS-E-MISLABBEG: Missing ');
  WRITE  (Error_Log, 'label block begin call. ');
  WRITELN (Error_Log, 'Call to PLabAdd or PLabEnd');
  WRITELN (Error_Log, 'without call to: PLabBegn.');
END;
PSE_MisVecBeg:
BEGIN
  WRITE  (Error_Log, 'PS-E-MISVECBEG: Missing ');
  WRITE  (Error_Log, 'vector list begin call. ');
  WRITELN (Error_Log, 'Call to PVecList or PVecEnd');
  WRITELN (Error_log, 'without call to: PVecBegn.');
END;
PSE_NulNam:
BEGIN
  WRITE  (Error_Log, 'PS-E-NULNAM: Null name ');
  WRITELN (Error_Log, 'parameter is not allowed.');
END;
PSE_BadComTyp:
BEGIN
  WRITE  (Error_Log, 'PS-E-BADCOMTYP: Bad ');
  WRITE  (Error_Log, 'comparison type operator ');
  WRITELN (Error_Log, 'specified in call to:');
  WRITELN (Error_Log, 'PIfLevel.');
END;
PSE_InvFunNam:
BEGIN
  WRITE  (Error_Log, 'PS-E-INVFUNNAM: Invalid ');
  WRITE  (Error_Log, 'function name. Attempted PS ');
  WRITELN (Error_Log, '300 function instance failed');
  WRITE  (Error_Log, 'because the named function ');
  WRITE  (Error_Log, 'cannot possibly exist. The ');
  WRITELN (Error_Log, 'function name identifying the');
  WRITE  (Error_Log, 'function type to instance ');
  WRITE  (Error_Log, 'was longer than 256 ');
  WRITELN (Error_Log, 'characters.');
END;
PSE_NulNamReq:
BEGIN
  WRITE  (Error_Log, 'PS-E-NULNAMREQ: Null name ');
  WRITE  (Error_Log, 'parameter is required in ');
  WRITELN (Error_Log, 'operate node call following');
  WRITE  (Error_Log, 'a PPref or PFoll procedure ');
  WRITELN (Error_Log, 'call.');
END;
```

```
  PSE_TooManEnd:
  BEGIN
    WRITE   (Error_Log, 'PS-E-TOOMANEND:  Too many ');
    WRITELN (Error_Log, 'END_STRUCTURE calls invoked.');
  END;
  PSE_NotAtt:
  BEGIN
    WRITE   (Error_Log, 'PS-E-NOTATT:  The PS 300 ');
    WRITE   (Error_Log, 'communications link has not ');
    WRITELN (Error_Log, 'yet been established.');
    WRITE   (Error_Log, 'PAttach has not been called ');
    WRITELN (Error_Log, 'or failed.');
  END;
  PSE_OveDurRea:
  BEGIN
    WRITE   (Error_Log, 'PS-E-OVEDURREA:  An overrun ');
    WRITE   (Error_Log, 'occurred during a read ');
    WRITELN (Error_Log, 'operation.');
    WRITE   (Error_Log, 'The specified input buffer ');
    WRITE   (Error_Log, 'in call to:  PGET  or:  PGETW');
    WRITELN (Error_Log, ' was too small and truncation');
    WRITELN (Error_Log, 'has occurred.');
  END;
  PSE_PhyDevTyp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-PHYDEVTYP:  Missing or ');
    WRITE   (Error_Log, 'invalid physical device type ');
    WRITELN (Error_Log, 'specifier in call to PAttach.');
    VAX_Specific;
  END;
  PSE_LogDevNam:
  BEGIN
    WRITE   (Error_Log, 'PS-E-LOGDEVNAM:  Missing or ');
    WRITE   (Error_Log, 'invalid logical device name ');
    WRITELN (Error_Log, 'specifier in call to PAttach.');
    VAX_Specific;
  END;
  PSE_AttDelExp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-ATTDELEXP:  Attach ');
    WRITE   (Error_Log, 'parameter string delimiter ');
    WRITELN (Error_Log, '"/" was expected.');
    VAX_Specific;
  END;
  OTHERWISE UnknownError;
END
```

```
{ Identify fatal errors }

ELSE Case Error_Code OF
  PSF_PhyAttFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYATTFAI: Physical ');
    WRITELN (Error_Log, 'attach operation failed.');
  END;
  PSF_PhyDetFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYDETFAI: Physical ');
    WRITELN (Error_Log, 'detach operation failed.');
  END;
  PSF_PhyGetFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYGETFAI: Physical ');
    WRITELN (Error_Log, 'get operation failed.');
  END;
  PSF_PhyPutFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYPUTFAI: Physical ');
    WRITELN (Error_Log, 'put operation failed.');
  END;
  PSF_BufTooLar:
  BEGIN
    WRITE   (Error_Log, 'PS-F-BUFTOOLAR: Buffer too ');
    WRITE   (Error_Log, 'large error in call to: ');
    WRITELN (Error_Log, 'PSPUT.');
    WRITE   (Error_Log, 'This error should NEVER ');
    WRITE   (Error_Log, 'occur and indicates a ');
    WRITELN (Error_Log, 'Procedural Interface (GSR)');
    WRITELN (Error_Log, 'validity check.');
    VAX_Specific;
  END;
  PSF_WroNumArg:
  BEGIN
    WRITE   (Error_Log, 'PS-F-WRONUMARG: Wrong ');
    WRITE   (Error_Log, 'number of arguments in call ');
    WRITELN (Error_Log, 'to Procedural Interface (GSR)');
    WRITE   (Error_Log, 'low-level I/O procedure ');
    WRITELN (Error_Log, '(source file: PROIOLIB.MAR).');
    WRITE   (Error_Log, 'This error should NEVER ');
    WRITE   (Error_Log, 'occur and indicates a ');
    WRITELN (Error_Log, 'Procedural Interface (GSR) ');
    WRITELN (Error_Log, 'validity check.');
    VAX_Specific;
  END;
```

```
    PSF_ProTooLar:
    BEGIN
      WRITE   (Error_Log, 'PS-F-PROTOOLAR: Prompt ');
      WRITE   (Error_Log, 'buffer too large error in ');
      WRITELN (Error_Log, 'call to: PSPRCV.');
      WRITE   (Error_Log, 'This error should NEVER ');
      WRITE   (Error_Log, 'occur and indicates a ');
      WRITELN (Error_Log, 'Procedural Interface (GSR) ');
      WRITELN (Error_Log, 'validity check.');
      VAX_Specific;
    END;
    OTHERWISE UnknownError;
  END;

  IF (Error_code >= PSF_PhyAttFai) AND
     (Error_code <= PSF_PhyPutFai) THEN BEGIN
    Pspvmserr ( VMSdef, PIdef );
    WRITELN (Error_Log, 'DEC VAX/VMS Error definition is:');
    WRITELN (Error_Log,  VMSdef );
    WRITE   (Error_Log, 'Procedural Interface (GSR) ');
    WRITE   (Error_Log, 'Interpretation of ');
    WRITELN (Error_Log, 'DEC VAX/VMS completion code:');
    WRITELN (Error_Log,  PIdef );
    WRITE   (Error_Log, 'DEC VAX/VMS Error code value ');
    WRITELN (Error_Log, 'was: ', Psvmserr );
  END;
  WRITELN (Error_Log);
END;


PROCEDURE DetachErrorHan (Detach_Error : INTEGER);
BEGIN
  WRITE   (Error_Log, 'PS-I-ERRWARDET: Error/warning ');
  WRITE   (Error_Log, 'trying to Detach ');
  WRITELN (Error_Log, 'the communications link between ');
  WRITELN (Error_Log, 'the PS 300 and the host.');
  IdentifyCompletionCode (Detach_Error);
END;


BEGIN
  IF NOT ErrorFileOpen THEN BEGIN

    { Open error file for the logging of errors }

    OPEN (Error_Log, 'Proerror.log', History := NEW);
    REWRITE (Error_Log);
    ErrorFileOpen := TRUE;
  END;
```

```
  IdentifyCompletionCode (Error_Code);
  IF Error_code >= 512 THEN BEGIN
    WRITE  (Error_Log, 'PS-I-ATDCOMLNK:  Attempting ');
    WRITE  (Error_Log, 'to detach PS 300');
    WRITELN (Error_Log, '/Host communications link.');

    { Use different error handler so as          }
    { not to get caught in a recursive           }
    { loop if we consistently get an             }
    { error when attempting to detach            }

    PDetach (DetachErrorHan);
    CLOSE (Error_Log);
    IF (Error_code >= PSF_PhyAttFai) AND
      (Error_code <= PSF_PhyPutFai)
     { identify VMS error if there was one }

     THEN LIB$STOP (PsVMSerr)
     ELSE HALT;
  END;
END;



FUNCTION Uppercase (Chara : CHAR) : CHAR;
BEGIN
  IF (Chara >= 'a') AND (Chara <= 'z')
    THEN Uppercase := CHR (ORD (Chara) - 32)
    ELSE Uppercase := Chara;
END;



PROCEDURE Attach;

VAR
  DeviceSpec : CHAR;
  DeviceName : VARYING [5] OF CHAR;
  AttachParm : P_VaryingType;

BEGIN
  DeviceSpec := ' ';
  REPEAT
    IF DeviceSpec <> ' ' THEN
      WRITELN (OUTPUT, 'Invalid device type specified.');
    WRITE  (OUTPUT, 'Device Interface type = (PARALLEL, ');
    WRITE  (OUTPUT, 'DMR-11, Asynchronous): _');
    IF EOLN (INPUT)
      THEN DeviceSpec := ' '
```

```
    ELSE DeviceSpec := Uppercase (INPUT¢);
   READLN (INPUT);
  UNTIL (DeviceSpec = 'P') OR (DeviceSpec = 'D') OR
     (DeviceSpec = 'A');
  REPEAT
   WRITE  (OUTPUT, 'Physical device name (i.e. ');
   WRITE  (OUTPUT, 'TT, TTA6, XMD0): _');
   READLN (INPUT,  DeviceName);
  UNTIL LENGTH (DeviceName) > 0;
  AttachParm := 'Logdevnam=' + DeviceName + ':/Phydevtyp=';
  IF Uppercase (DeviceSpec) = 'P'
   THEN AttachParm := AttachParm + 'PARALLEL'
   ELSE IF Uppercase (DeviceSpec) = 'D'
        THEN AttachParm := AttachParm + 'DMR-11'
        ELSE AttachParm := AttachParm + 'Async';
  Pattach (AttachParm, ERR);
END;


PROCEDURE Computename (   NameId : INTEGER;
                VAR Name   : P_VaryingType);
VAR
 j    : INTEGER;

BEGIN
 Name := 'List000';
 j := 7;
 WHILE (NameId > 0) DO BEGIN
  Name [j] := CHR (NameId MOD 10 + ORD ('0'));
  NameId := NameId DIV 10;
  j := PRED (j);
 END;
END;


PROCEDURE ComputeWave (   Theta   : REAL;
                VAR VecList : P_VectorListType);
CONST
 Amp       = 0.8;
 Alpha     = -0.02;
 Beta      = 0.2513274123;

VAR
 i        : INTEGER;
 Addr     : INTEGER;
 Iaddr    : INTEGER;
```

```
  BEGIN
    Iaddr := 0;
    FOR i := 0 TO 49 DO BEGIN
      Iaddr := SUCC (Iaddr);
      VecList [Iaddr].V4 [1] := i / 50.0;
      VecList [Iaddr].V4 [2] := Amp * EXP (Alpha * i)
                              * COS (Theta - Beta * i);
      VecList [Iaddr].V4 [3] := 0;
      VecList [Iaddr].V4 [4] := 1 - i/150.0;
      VecList [Iaddr].Draw := TRUE;
      Iaddr := SUCC (Iaddr);
      VecList [Iaddr].V4 [1] := VecList [PRED (Iaddr)].V4 [1];
      VecList [Iaddr].V4 [2] := 0;
      VecList [Iaddr].V4 [3] := 0.5;
      VecList [Iaddr].V4 [4] := VecList [PRED (Iaddr)].V4 [4];
      VecList [Iaddr].Draw := TRUE;
    END;
  END;

BEGIN
  Attach;                    { Do the Attach }
  At.V4 [1] := 0.3;
  At.V4 [2] := 0;
  At.V4 [3] := 0;
  From.V4 [1] := 0;
  From.V4 [2] := 0;
  From.V4 [3] := -1;
  Up.V4 [1] := 0.3;
  Up.V4 [2] := 1;
  Up.V4 [3] := 0;
  Y_Up.V4 [1] := 0;
  Y_Up.V4 [2] := 1;
  Y_Up.V4 [3] := 0;
  Zero_vec.V4 [1] := 0;
  Zero_vec.V4 [2] := 0;
  Zero_vec.V4 [3] := 0;
  PInit   ( Err );
  PEyeBack ( 'eye', 1.0, 0.0, 0.0, 2.0, 0.0,
           1000.0, 'inten', Err );
  PSetInt  ( 'inten', TRUE, 0.5, 1.0, 'look', Err );
  PLookat  ( 'look', At, From, Up, 'pic', Err );
  PFnInst  ( 'atx', 'xvec', Err );
  PFnInst  ( 'aty', 'yvec', Err );
  PFnInst  ( 'atz', 'zvec', Err );
  PFnInst  ( 'fromx', 'xvec', Err );
  PFnInst  ( 'fromy', 'yvec', Err );
  PFnInst  ( 'fromz', 'zvec', Err );
  PFnInst  ( 'ac_at', 'accumulate', Err );
  PFnInst  ( 'ac_from', 'accumulate', Err );
```

```
PFnInst  ( 'add_up', 'addc', Err );
PFnInstN ( 'sync_up', 'sync', 3, Err );
PFnInst  ( 'fix_sync', 'nop', Err );
PConnect ( 'sync_up', 3, 1, 'fix_sync', Err );
PConnect ( 'fix_sync', 1, 3, 'sync_up', Err );
PSndBool ( TRUE, 3, 'sync_up', Err );
PFnInst  ( 'look_at', 'lookat', Err );
PConnect ( 'dials', 1, 1, 'atx', Err );
PConnect ( 'dials', 2, 1, 'aty', Err );
PConnect ( 'dials', 3, 1, 'atz', Err );
PConnect ( 'dials', 5, 1, 'fromx', Err );
PConnect ( 'dials', 6, 1, 'fromy', Err );
PConnect ( 'dials', 7, 1, 'fromz', Err );
PConnect ( 'atx', 1, 1, 'ac_at', Err );
PConnect ( 'aty', 1, 1, 'ac_at', Err );
PConnect ( 'atz', 1, 1, 'ac_at', Err );
PConnect ( 'fromx', 1, 1, 'ac_from', Err );
PConnect ( 'fromy', 1, 1, 'ac_from', Err );
PConnect ( 'fromz', 1, 1, 'ac_from', Err );
PConnect ( 'ac_at', 1, 1, 'sync_up', Err );
PConnect ( 'ac_at', 1, 1, 'add_up',  Err );
PConnect ( 'add_up',1, 2, 'sync_up', Err );
PConnect ( 'sync_up', 1, 1, 'look_at', Err );
PConnect ( 'sync_up', 2, 3, 'look_at', Err );
PConnect ( 'ac_from', 1, 2, 'look_at', Err );
PSndV3D  ( At, 2, 'ac_at', Err );
PSndV3D  ( From, 2, 'ac_from', Err );
PSndV3D  ( Y_up, 2, 'add_up', Err );
PConnect ( 'look_at', 1, 1, 'look', Err );
PFnInst  ( 'fix_at', 'const', Err );
PConnect ( 'ac_from', 1, 1, 'fix_at', Err );
PConnect ( 'fix_at', 1, 1, 'ac_at', Err );
PSndV3D  ( Zero_vec, 2, 'fix_at', Err );
PSndV3D  ( Zero_vec, 1, 'ac_from', Err );
PInst    ( 'pic', '', Err );
Dtheta := 10.0 * Deg_rad;
Theta  := -Dtheta;
FOR i := 1 TO 36 DO BEGIN
  Theta := Theta + Dtheta;
  Computewave (Theta, Vecs);
  FOR k := 1 TO 50 DO BEGIN
    FOR l := 1 TO 4 DO Front [k].V4 [l]
      := Vecs [SUCC (PRED (k) * 2)].V4 [l];
    Front [k].Draw := Vecs [SUCC (PRED (k) * 2)].Draw;
  END;
  Computename ( i, Name );
  PBegins  ( Name, Err );
  PSetR    ( '', 1, 35, FALSE, i, '', Err );
```

```
    PIfPhase ( '', TRUE, '', Err );
    PVecBegn ( '', 100, FALSE, FALSE, 3, P_Sepa, Err );
    PVecList ( 100, Vecs, Err );
    PVecEnd  ( Err );
    PVecBegn ( '', 50, FALSE, FALSE, 3, P_Conn, Err );
    PVecList ( 50, Front, Err );
    PVecEnd  ( Err );
    PEnds    ( Err );
    PIncl    ( Name, 'pic', Err );
  END;
  PDisplay ( 'eye', Err );
  PSndStr  ( 'X', 1, 'Dlabel1', Err );
  PSndStr  ( 'Y', 1, 'Dlabel2', Err );
  PSndStr  ( 'Z', 1, 'Dlabel3', Err );
  PSndStr  ( 'Look At', 1, 'Dlabel4', Err );
  PSndStr  ( 'X', 1, 'Dlabel5', Err );
  PSndStr  ( 'Y', 1, 'Dlabel6', Err );
  PSndStr  ( 'Z', 1, 'Dlabel7', Err );
  PSndStr  ( 'From', 1, 'Dlabel8', Err );
  Pdetach  ( Err );
END.
```

This is a Pascal vector list example program.

```
PROGRAM CircleTest (INPUT, OUTPUT);

CONST
  %INCLUDE 'PROCONST.PAS'

TYPE
  %INCLUDE 'PROTYPES.PAS'

VAR
  circle_list        : P_VectorListType;
  Dimensionality     : INTEGER;
  Class              : INTEGER;
  ClassType          : CHAR;
  Mode               : CHAR;
  BlockNormalized    : BOOLEAN;
```

```
%INCLUDE 'PROEXTRN.PAS'


{ The following Error Handler demonstrates the                  }
{ general overall recommended form that the user's              }
{ own error handler should follow.                              }
{                                                               }
{ This error handler upon being invoked writes ALL              }
{ messages to the data file: 'PROERROR.LOG' for 2               }
{ reasons:                                                      }
{                                                               }
{    1. The error handler should NOT immediately                }
{       write information out on the PS 300 screen              }
{       since the explanatory text defining the                 }
{       error or warning condition may be taken as              }
{       data by the PS 300 and therefore wind up                }
{       not being displayed on the PS 300 screen                }
{       (as in the case of a catastrophic data                  }
{       transmission error).                                    }
{                                                               }
{    2. The logging of errors and warnings to a                 }
{       logfile allows any errors and/or warnings               }
{       to be reviewed at a later time.                         }


PROCEDURE Err ( Error_code: Integer );

VAR
  VMSdef, PIdef : P_VaryingType;
  Error_Log     : [STATIC] TEXT;
  ErrorFileOpen : [STATIC] BOOLEAN := FALSE;


  [EXTERNAL] PROCEDURE LIB$STOP
          (%IMMED CompletionCode : INTEGER); EXTERN;


  PROCEDURE IBM_Specific;
  BEGIN
    WRITE   (Error_Log, 'This error/warning is ');
    WRITE   (Error_Log, 'applicable ONLY to the IBM ');
    WRITELN (Error_Log, 'version of the');
    WRITELN (Error_Log, 'Procedural Interface (GSR).');
  END;
```

```
PROCEDURE VAX_Specific;
BEGIN
  WRITE   (Error_Log, 'This error/warning is ');
  WRITE   (Error_Log, 'applicable ONLY to the DEC ');
  WRITELN (Error_Log, 'VAX/VMS version of');
  WRITE   (Error_Log, 'the Procedural Interface ');
  WRITELN (Error_Log, '(GSR).');
END;


PROCEDURE UnknownError;
BEGIN
  WRITE   (Error_Log, 'PS-W-UNRCOMCOD:  ');
  WRITE   (Error_Log, 'Procedural Interface ');
  WRITE   (Error_Log, '(GSR) completion ');
  IF Error_code < 512
    THEN WRITE (Error_Log, 'warning ')
    ELSE IF Error_code < 1024
         THEN WRITE (Error_Log, 'error ')
         ELSE WRITE (Error_Log, 'fatal error ');
  WRITELN (Error_Log, 'code is unrecognized.');
  WRITE   (Error_Log, 'Probable Procedural ');
  WRITE   (Error_Log, 'Interface (GSR) Internal ');
  WRITELN (Error_Log, 'validity check error.');
END;

PROCEDURE IdentifyCompletionCode
       (Error_code : INTEGER);
BEGIN
  WRITE   (Error_Log, 'PS-I-PROERRWAR:  Procedural ');
  WRITE   (Error_Log, 'Interface (GSR) warning/');
  WRITE   (Error_Log, 'error completion code was ');
  WRITELN (Error_Log, 'received.');

  { Identify warning codes }

  IF Error_Code < 512 THEN CASE Error_Code OF
    PSW_BadNamChr:
    BEGIN
      WRITE   (Error_Log, 'PS-W-BADNAMCHR:  Bad ');
      WRITE   (Error_Log, 'character in name was ');
      WRITELN (Error_Log, 'translated to:  "_".');
    END;
    PSW_NamTooLon:
    BEGIN
      WRITE   (Error_Log, 'PS-W-NAMTOOLON:  Name too ');
      WRITE   (Error_Log, 'long.  Name was truncated to ');
      WRITELN (Error_Log, '256 characters.');
    END;
```

```
PSW_StrTooLon:
BEGIN
  WRITE  (Error_Log, 'PS-W-STRTOOLON: String too ');
  WRITE  (Error_Log, 'long. String was truncated ');
  WRITELN (Error_Log, 'to 240 characters.');
END;
PSW_AttAlrDon:
BEGIN
  WRITE  (Error_Log, 'PS-W-ATTALRDON: Attach ');
  WRITE  (Error_Log, 'already done. Multiple call ');
  WRITELN (Error_Log, 'to PAttach without');
  WRITE  (Error_Log, 'intervening PDetach call '); .
  WRITELN (Error_Log, 'ignored.');
END;
PSW_AtnKeySee:
BEGIN
  WRITE  (Error_Log, 'PS-W-ATNKEYSEE: Attention ');
  WRITELN (Error_Log, 'key seen (depressed).');
  IBM_Specific;
END;
PSW_BadGenChr:
BEGIN
  WRITE  (Error_Log, 'PS-W-BADGENCHR: Bad generic ');
  WRITE  (Error_Log, 'channel character. Bad ');
  WRITELN (Error_Log, 'character in string sent via:');
  WRITE  (Error_Log, ' PPutGX  was translated to ');
  WRITELN (Error_Log, 'a blank.');
  IBM_Specific;
END;
PSW_BadStrChr:
BEGIN
  WRITE  (Error_Log, 'PS-W-BADSTRCHR: Bad ');
  WRITE  (Error_Log, 'character in string was ');
  WRITELN (Error_Log, 'translated to a blank.');
  IBM_Specific;
END;
PSW_BadParChr:
BEGIN
  WRITE  (Error_Log, 'PS-W-BADPARCHR: Bad parser ');
  WRITE  (Error_Log, 'channel character. Bad ');
  WRITELN (Error_Log, 'character in string sent to');
  WRITE  (Error_Log, 'PS 300 parser via: PPutP  ');
  WRITELN (Error_Log, 'was translated to a blank.');
  IBM_Specific;
END;
OTHERWISE UnknownError;
END
```

```
{ Identify errors }

ELSE IF Error_code < 1024 THEN CASE Error_Code OF
 PSE_InvMuxCha:
 BEGIN
  WRITE  (Error_Log, 'PS-E-INVMUXCHA: Invalid ');
  WRITE  (Error_Log, 'multiplexing channel ');
  WRITELN (Error_Log, 'specified in call to:');
  WRITELN (Error_Log, 'PMuxCI, PMuxP, or PMuxG.');
 END;
 PSE_InvVecCla:
 BEGIN
  WRITE  (Error_Log, 'PS-E-INVVECCLA: Invalid ');
  WRITE  (Error_Log, 'vector list class specified ');
  WRITELN (Error_Log, 'in call to: PVecBegn.');
 END;
 PSE_InvVecDim:
 BEGIN
  WRITE  (Error_Log, 'PS-E-INVVECDIM: Invalid ');
  WRITE  (Error_Log, 'vector list dimension ');
  WRITELN (Error_Log, 'specified in call to');
  WRITELN (Error_Log, 'PVecBegn.');
 END;
 PSE_PreOpeExp:
 BEGIN
  WRITE  (Error_Log, 'PS-E-PREOPEEXP: Prefix ');
  WRITELN (Error_Log, 'operator call was expected.');
 END;
 PSE_FolOpeExp:
 BEGIN
  WRITE  (Error_Log, 'PS-E-FOLOPEEXP: Follow ');
  WRITELN (Error_Log, 'operator call was expected.');
 END;
 PSE_LabBlkExp:
 BEGIN
  WRITE  (Error_Log, 'PS-E-LABBLKEXP: Call to ');
  WRITE  (Error_Log, 'PLabAdd or PLabEnd was ');
  WRITELN (Error_Log, 'expected.');
 END;
 PSE_VecLisExp:
 BEGIN
  WRITE  (Error_Log, 'PS-E-VECLISEXP: Call to ');
  WRITE  (Error_Log, 'PVecList or PVecEnd was ');
  WRITELN (Error_Log, 'expected.');
 END;
```

```
PSE_AttMulVec:
BEGIN
  WRITE   (Error_Log, 'PS-E-ATTMULVEC: Attempted ');
  WRITE   (Error_Log, 'multiple call sequence to ');
  WRITELN (Error_Log, 'PVecList is NOT permitted');
  WRITELN (Error_Log, 'for BLOCK normalized vectors.');
END;
PSE_MisLabBeg:
BEGIN
  WRITE   (Error_Log, 'PS-E-MISLABBEG: Missing ');
  WRITE   (Error_Log, 'label block begin call. ');
  WRITELN (Error_Log, 'Call to PLabAdd or PLabEnd');
  WRITELN (Error_Log, 'without call to: PLabBegn.');
END;
PSE_MisVecBeg:
BEGIN
  WRITE   (Error_Log, 'PS-E-MISVECBEG: Missing ');
  WRITE   (Error_Log, 'vector list begin call. ');
  WRITELN (Error_Log, 'Call to PVecList or PVecEnd');
  WRITELN (Error_log, 'without call to: PVecBegn.');
END;
PSE_NulNam:
BEGIN
  WRITE   (Error_Log, 'PS-E-NULNAM: Null name ');
  WRITELN (Error_Log, 'parameter is not allowed.');
END;
PSE_BadComTyp:
BEGIN
  WRITE   (Error_Log, 'PS-E-BADCOMTYP: Bad ');
  WRITE   (Error_Log, 'comparison type operator ');
  WRITELN (Error_Log, 'specified in call to:');
  WRITELN (Error_Log, 'PIfLevel.');
END;
PSE_InvFunNam:
BEGIN
  WRITE   (Error_Log, 'PS-E-INVFUNNAM: Invalid ');
  WRITE   (Error_Log, 'function name. Attempted PS ');
  WRITELN (Error_Log, '300 function instance failed');
  WRITE   (Error_Log, 'because the named function ');
  WRITE   (Error_Log, 'cannot possibly exist. The ');
  WRITELN (Error_Log, 'function name identifying the');
  WRITE   (Error_Log, 'function type to instance ');
  WRITE   (Error_Log, 'was longer than 256 ');
  WRITELN (Error_Log, 'characters.');
END;
```

```
PSE_NulNamReq:
BEGIN
  WRITE  (Error_Log, 'PS-E-NULNAMREQ: Null name ');
  WRITE  (Error_Log, 'parameter is required in ');
  WRITELN (Error_Log, 'operate node call following');
  WRITE  (Error_Log, 'a PPref or PFoll procedure ');
  WRITELN (Error_Log, 'call.');
END;
PSE_TooManEnd:
BEGIN
  WRITE  (Error_Log, 'PS-E-TOOMANEND: Too many ');
  WRITELN (Error_Log, 'END_STRUCTURE calls invoked.');
END;
PSE_NotAtt:
BEGIN
  WRITE  (Error_Log, 'PS-E-NOTATT: The PS 300 ');
  WRITE  (Error_Log, 'communications link has not ');
  WRITELN (Error_Log, 'yet been established.');
  WRITE  (Error_Log, 'PAttach has not been called ');
  WRITELN (Error_Log, 'or failed.');
END;
PSE_OveDurRea:
BEGIN
  WRITE  (Error_Log, 'PS-E-OVEDURREA: An overrun ');
  WRITE  (Error_Log, 'occurred during a read ');
  WRITELN (Error_Log, 'operation.');
  WRITE  (Error_Log, 'The specified input buffer ');
  WRITE  (Error_Log, 'in call to: PGET or: PGETW');
  WRITELN (Error_Log, ' was too small and truncation');
  WRITELN (Error_Log, 'has occurred.');
END;
PSE_PhyDevTyp:
BEGIN
  WRITE  (Error_Log, 'PS-E-PHYDEVTYP: Missing or ');
  WRITE  (Error_Log, 'invalid physical device type ');
  WRITELN (Error_Log, 'specifier in call to PAttach.');
  VAX_Specific;
END;
PSE_LogDevNam:
BEGIN
  WRITE  (Error_Log, 'PS-E-LOGDEVNAM: Missing or ');
  WRITE  (Error_Log, 'invalid logical device name ');
  WRITELN (Error_Log, 'specifier in call to PAttach.');
  VAX_Specific;
END;
```

```
  PSE_AttDelExp:
  BEGIN
    WRITE   (Error_Log, 'PS-E-ATTDELEXP:  Attach ');
    WRITE   (Error_Log, 'parameter string delimiter ');
    WRITELN (Error_Log, '"/" was expected.');
    VAX_Specific;
  END;
  OTHERWISE UnknownError;
END

{ Identify fatal errors }

ELSE Case Error_Code OF
  PSF_PhyAttFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYATTFAI:  Physical ');
    WRITELN (Error_Log, 'attach operation failed.');
  END;
  PSF_PhyDetFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYDETFAI:  Physical ');
    WRITELN (Error_Log, 'detach operation failed.');
  END;
  PSF_PhyGetFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYGETFAI:  Physical ');
    WRITELN (Error_Log, 'get operation failed.');
  END;
  PSF_PhyPutFai:
  BEGIN
    WRITE   (Error_Log, 'PS-F-PHYPUTFAI:  Physical ');
    WRITELN (Error_Log, 'put operation failed.');
  END;
  PSF_BufTooLar:
  BEGIN
    WRITE   (Error_Log, 'PS-F-BUFTOOLAR:  Buffer too ');
    WRITE   (Error_Log, 'large error in call to:  ');
    WRITELN (Error_Log, 'PSPUT.');
    WRITE   (Error_Log, 'This error should NEVER ');
    WRITE   (Error_Log, 'occur and indicates a ');
    WRITELN (Error_Log, 'Procedural Interface (GSR)');
    WRITELN (Error_Log, 'validity check.');
    VAX_Specific;
  END;
```

```
    PSF_WroNumArg:
    BEGIN
      WRITE   (Error_Log, 'PS-F-WRONUMARG: Wrong ');
      WRITE   (Error_Log, 'number of arguments in call ');
      WRITELN (Error_Log, 'to Procedural Interface (GSR)');
      WRITE   (Error_Log, 'low-level I/O procedure ');
      WRITELN (Error_Log, '(source file: PROIOLIB.MAR).');
      WRITE   (Error_Log, 'This error should NEVER ');
      WRITE   (Error_Log, 'occur and indicates a ');
      WRITELN (Error_Log, 'Procedural Interface (GSR) ');
      WRITELN (Error_Log, 'validity check.');
      VAX_Specific;
    END;
    PSF_ProTooLar:
    BEGIN
      WRITE   (Error_Log, 'PS-F-PROTOOLAR: Prompt ');
      WRITE   (Error_Log, 'buffer too large error in ');
      WRITELN (Error_Log, 'call to: PSPRCV.');
      WRITE   (Error_Log, 'This error should NEVER ');
      WRITE   (Error_Log, 'occur and indicates a ');
      WRITELN (Error_Log, 'Procedural Interface (GSR) ');
      WRITELN (Error_Log, 'validity check.');
      VAX_Specific;
    END;
    OTHERWISE UnknownError;
  END;
  IF (Error_code >= PSF_PhyAttFai) AND
     (Error_code <= PSF_PhyPutFai) THEN BEGIN
    Pspvmserr ( VMSdef, PIdef );
    WRITELN (Error_Log, 'DEC VAX/VMS Error definition is:');
    WRITELN (Error_Log,  VMSdef );
    WRITE   (Error_Log, 'Procedural Interface (GSR) ');
    WRITE   (Error_Log, 'Interpretation of ');
    WRITELN (Error_Log, 'DEC VAX/VMS completion code:');
    WRITELN (Error_Log,  PIdef );
    WRITE   (Error_Log, 'DEC VAX/VMS Error code value ');
    WRITELN (Error_Log, 'was: ', Psvmserr );
  END;
  WRITELN (Error_Log);
END;


PROCEDURE DetachErrorHan (Detach_Error : INTEGER);
BEGIN
  WRITE   (Error_Log, 'PS-I-ERRWARDET: Error/warning ');
  WRITE   (Error_Log, 'trying to Detach ');
  WRITELN (Error_Log, 'the communications link between ');
  WRITELN (Error_Log, 'the PS 300 and the host.');
  IdentifyCompletionCode (Detach_Error);
END;
```

```
BEGIN
 IF NOT ErrorFileOpen THEN BEGIN

   { Open error file for the logging of errors }

   OPEN (Error_Log, 'Proerror.log', History := NEW);
   REWRITE (Error_Log);
   ErrorFileOpen := TRUE;
 END;
 IdentifyCompletionCode (Error_Code);
 IF Error_code >= 512 THEN BEGIN
   WRITE   (Error_Log, 'PS-I-ATDCOMLNK:  Attempting ');
   WRITE   (Error_Log, 'to detach PS 300');
   WRITELN (Error_Log, '/Host communications link.');

   { Use different error handler so as }
   { not to get caught in a recursive  }
   { loop if we consistently get an    }
   { error when attempting to detach   }

   PDetach (DetachErrorHan);
   CLOSE (Error_Log);
   IF (Error_code >= PSF_PhyAttFai) AND
     (Error_code <= PSF_PhyPutFai)

    { identify VMS error if there was one }

    THEN LIB$STOP (PsVMSerr)
    ELSE HALT;
 END;
END;


FUNCTION Uppercase (Chara : CHAR) : CHAR;
BEGIN
 IF (Chara >= 'a') AND (Chara <= 'z')
   THEN Uppercase := CHR (ORD (Chara) - 32)
   ELSE Uppercase := Chara;
END;


PROCEDURE Circle;

CONST
 Deg_rad    = 0.017453292;
```

```
VAR
  Theta  : REAL;
  DTheta  : REAL;
  i      : INTEGER;
  Draw    : BOOLEAN;

BEGIN
  Draw := FALSE;
  DTheta := 3.6 * Deg_Rad;
  Theta  := 0;
  FOR i := 1 TO 101 DO BEGIN
    circle_list [i].v4 [1] := 0.8 * cos ( theta );
    circle_list [i].v4 [2] := 0.8 * sin ( theta );
    circle_list [i].v4 [3] := 0;
    circle_list [i].v4 [4] := 1;
    circle_list [i].Draw := Draw;
    Theta := Theta + DTheta;
    Draw := NOT Draw;
  END;
END;


PROCEDURE Attach;

VAR
  DeviceSpec : CHAR;
  DeviceName : VARYING [5] OF CHAR;
  AttachParm : P_VaryingType;

BEGIN
  DeviceSpec := ' ';
  REPEAT
    IF DeviceSpec <> ' ' THEN
      WRITELN (OUTPUT, 'Invalid device type specified.');
    WRITE  (OUTPUT, 'Device Interface type = (PARALLEL, ');
    WRITE  (OUTPUT, 'DMR-11, Asynchronous): _');
    IF EOLN (INPUT)
      THEN DeviceSpec := ' '
      ELSE DeviceSpec := Uppercase (INPUT¢);
    READLN (INPUT);
  UNTIL (DeviceSpec = 'P') OR (DeviceSpec = 'D') OR
      (DeviceSpec = 'A');
  REPEAT
    WRITE  (OUTPUT, 'Physical device name (i.e. ');
    WRITE  (OUTPUT, 'TT, TTA6, XMD0): _');
    READLN (INPUT,  DeviceName);
  UNTIL LENGTH (DeviceName) > 0;
```

```
    AttachParm := 'Logdevnam=' + DeviceName + ':/Phydevtyp=';
    IF Uppercase (DeviceSpec) = 'P'
      THEN AttachParm := AttachParm + 'PARALLEL'
      ELSE IF Uppercase (DeviceSpec) = 'D'
           THEN AttachParm := AttachParm + 'DMR-11'
           ELSE AttachParm := AttachParm + 'Async';
    Pattach (AttachParm, ERR);
  END;

BEGIN
  WRITE  (OUTPUT, 'Vector mode = (Block, Vector): _');
  READLN  (INPUT, Mode);
  IF Uppercase (Mode) = 'B'
    THEN BlockNormalized := TRUE
    ELSE BlockNormalized := FALSE;
  WRITE  (OUTPUT, 'Dimensionality = (2, 3): _');
  READLN  (INPUT,  Dimensionality);
  WRITE  (OUTPUT, 'Class = (Connected, Dots, Itemized, ');
  WRITE  (OUTPUT, 'Separate): _');
  READLN  (INPUT,  ClassType);
  CASE Uppercase (ClassType) OF
    'C' : Class := P_Conn;
    'D' : Class := P_Dots;
    'I' : Class := P_Item;
    'S' : Class := P_Sepa;
    OTHERWISE Class := P_Conn;
  END;
  Attach;
  Pinit ( Err );
  Circle;
  Pvecbegn ('circle', 101, BlockNormalized, FALSE,
          Dimensionality, Class, Err);
  Pveclist ( 101, circle_list, err );
  Pvecend ( err );
  pdisplay ('circle', err );
  Pdetach ( err );
END.
```

APPENDIX B. HOST_MESSAGE

This appendix contains the function network diagram and functional description of HOST_MESSAGE (an instance of the intrinsic function HOLD_MESSAGE) that supports the procedures PGetWait and PGET of the GSRs. This function is already part of the PS 300 system. When using the GSRs, all messages sent from the PS 300 to the host must be sent via this function.

The function HOST_MESSAGE is a F:NOP function directly connected to the function HOST_MESSAGEB. It is recommended that the user always send PS 300 output destined for the host computer to HOST_MESSAGE rather than HOST_MESSAGEB since the name of the latter function may change with a future release of runtime software.

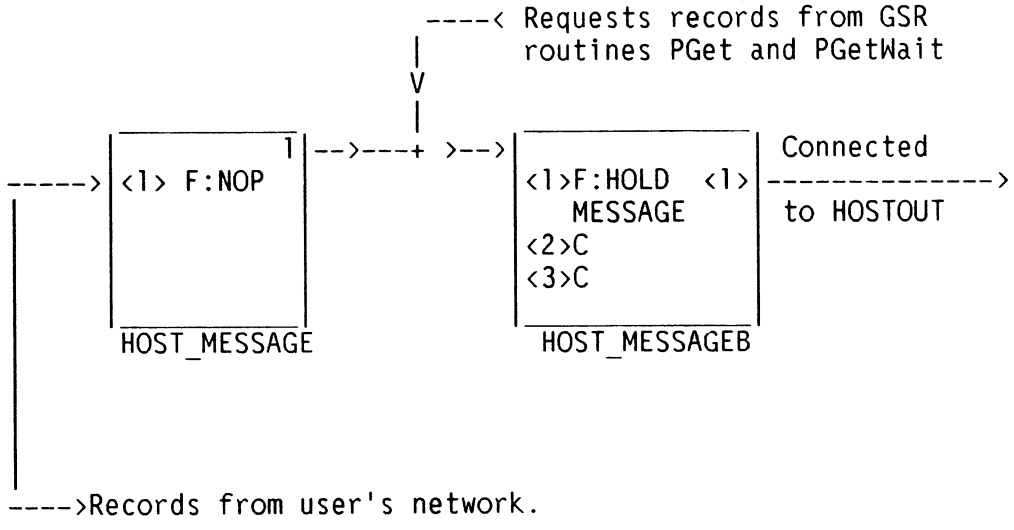

Figure B-1. Hold_Message Function Network Diagram

HOLD_MESSAGE:

INPUTS:

<1>: Qpackets of messages to be sent to the Host and Qintegers used to trigger the messages as follows:

FIX(0): Clear any messages waiting the FIFO queue of messages to be sent to the Host.

FIX(1): If a message is waiting, send it. Otherwise send the message indicating: "No-messages" as determined by input <3>.

FIX(2): If a message is waiting, send it. Otherwise, wait until a Qpacket message arrives on input <1> and then immediately send the message.

<2>C: Message Terminator Qpacket that is added to the end of messages arriving on input <1> just prior to transmission to the host.

The default input value for input <2> is a carriage return: CHR (13).

<3>C: "No-messages" Qpacket. If this function receives a FIX (1) on input <1>, then the message on this constant queue is sent ONLY if there are no other messages waiting to be sent on input <1>.

The default input value for input <3> is a carriage return: CHR (13).

OUTPUTS:

<1>: Qpacket sent to the Host Computer in response to the receipt of either a FIX (1) or FIX (2) on input <1>.

The GSR procedures: PGet and PGetWait specifically interrogate the function: HOST_MESSAGEB for input back to the host.

The procedure PGet is used to "poll" the PS 300 for data. If a message exists on the FIFO queue of HOST_MESSAGEB, then that message is removed from the queue and is returned by PGet. If no message was present in the input queue of HOST_MESSAGEB then the special: "No-messages" message as defined by input <3> of HOST_MESSAGE is returned.

The procedure PGetWait is similar in functionality to PGet with one important difference. PGetWait will NOT return to the caller until a message has been received from the PS 300. If no messages are present on the input queue of HOST_MESSAGEB, then the caller of PGetWait (Get message and wait for completion) will wait until a message is sent to input <1> of HOST_MESSAGEB.

NOTE

Messages received from the PS 300 via PGet and PGetWait may need to be "trimmed" of the trailing character(s) as defined by inputs <2> and <3> of HOST_MESSAGEB if either of them is changed from the default value of carriage return (Character 13). The DEC VAX/VMS PASCAL GSR will remove a single trailing carriage return from the message. Thus if a poll operation is requested and no messages are present, the GSR returns a zero-length message to the caller indicating that no messages were present because the default "No-message" message on input <3> of HOST_MESSAGEB is a carriage return. Similary, calls to PGetWait return the proper length. However, if the user chooses to change the HOST_MESSAGEB inputs <2> or <3>, then the user must compensate for any side effects so produced when calling PGet or PGetWait.

PS 300 DEC VAX/VMS FORTRAN-77 GRAPHICS SUPPORT ROUTINES

USER'S MANUAL

Supported Under PS 300 Graphics Firmware Release A1

CONTENTS

SECTION I

SECTION II

SECTION III

APPENDIX A. SAMPLE PROGRAMS

APPENDIX B. HOST_MESSAGE

INTRODUCTION

The PS 300 VAX FORTRAN-77 Graphics Support Routines (GSRs) are a package of FORTRAN subroutines that are executed on the host computer. These subroutines allow the host to communicate PS 300 commands directly to the PS 300 Command Interpreter. The GSRs provide subroutines for most commands acceptable by the PS 300 Graphics System.

The GSRs described here are written in FORTRAN-77 and require a FORTRAN-77 compiler to compile properly.

The purpose of this document is to provide a cross reference between the PS 300 command language and the corresponding FORTRAN subroutines of the GSRs.

This document should be used in conjunction with the **PS 300 Command Summary.** No attempt has been made in this document to provide tutorial information on the use of the PS 300 command language or syntax.

The GSRs are supported under PS 300 Graphics Firmware Release P5.V03 and higher. There are no specific hardware requirements.

This manual is divided into three sections. The first section is a guide to the GSRs. It contains information on the conventions and definitions used in the GSRs. There are several PS 300 commands that have not been implemented in the GSRs. These commands are documented under Excluded Commands.

A section titled Programming Suggestions has been provided that lists the GSR FORTRAN PARAMETER declarations that may be helpful to the user.

An error handling scheme has been employed to catch errors detected by the GSRs. A table of the error codes and definitions follows the listing of the Utility and Application Subroutines. Appendix A contains a sample error-handling subroutine.

The second section of the manual lists each GSR subroutine with its corresponding PS 300 command. The subroutines are presented in alphabetical order with parameters and the corresponding PS 300 command syntax.

When an example is given, it is shown with both the PS 300 command syntax and the subroutine parameters. Any notes following a subroutine describe discrepancies or restrictions that apply to the subroutine but may not apply to the corresponding PS 300 command.

An alphabetical listing of the PS 300 Commands, the corresponding FORTRAN subroutine, and the appropriate page reference is provided at the front of the description of the subroutines.

The third section contains the appendices. Appendix A contains sample programs that illustrate the use of the GSRs. Each program has an example of an error-handling subroutine. Appendix B contains a description of the PS 300 system function HOST_MESSAGE. Installation instructions are in the **System Manager Reference**, Volume 5 of this document set.

The GSRs were developed at Evans and Sutherland as a standard communication path between the PS 300 and the application program. Prior to this interface, communication with the PS 300 was supported by the Host Resident I/O Routines (PSIO). All commands were sent to the PS 300 as ASCII character strings (with the exception of vector lists). It was the responsibility of the application to format graphical information into the proper PS 300 commands. Typically, this was accomplished using FORTRAN ENCODE/DECODE and FORMAT statements, or equivalents, to build character strings to be sent to the Parser via PSSEND. PSVECS provided a faster communication path by formatting vector data into a "binary" format and including the proper routing information to bypass the Parser and communicate directly with the Command Interpreter.

The GSRs provide a set of subroutines that perform all formatting and routing duties for the application. They take advantage of the fact that all data formatting is performed by E&S supported code. The GSRs communicate nearly all commands directly to the Command Interpreter and achieve significant performance improvement over the ASCII form of the commands.

Applications

Typically, the subroutines will be used for the following applications:

- Attach to the graphics device
- Create and modify display structures
- Create, connect and modify function networks
- Receive data from the graphics device

Graphics Support Routines Conventions

The Graphics Support Routines make extensive use of the following data type definitions:

Boolean = Logical value true/false, generally LOGICAL*1.
Integer = Integer value always INTEGER*4.
Real = Real (floating point) number generally REAL*4.
String = Character string, CHARACTER*N.

For the FORTRAN version of the Graphics Support Routines, character strings require a delimiter character for length determination. Double quote (©"©) is the default delimiter. This delimiter may be changed using the PDELIM subroutine. A description of PDELIM is found in the Utility Subroutine section. The Graphics Support Routines use LEN (String) to determine the maximum length of a string. Therefore, if the delimiter is not specified, all characters up to LEN (String) will be used. Because of this, quoted strings may be used without delimiters, i.e. 'THIS' is treated the same as 'THIS"'.

Utility Subroutines

There are two types of supporting subroutines. Utility Subroutines are specific to the operation of the Graphics Support Routines. These calls are used to attach the PS 300, set the string delimiting character, select multiplexing channels, send and receive messages, and detach.

Application Subroutines

The Application Subroutines correspond almost one for one with the standard PS 300 Commands. Exceptions and exclusions are given following the text on the Application Subroutines.

In most cases, the names for the Application Subroutines were derived by choosing an abbreviation of the PS 300 commands and prefixing it with a P. Parameter ordering generally coincides with the PS 300 commands as well.

Examples of some of the Application Subroutines are shown below.

Example 1

For commands which build operate display structures, such as

Name:= operate parameter1,parameter2,..., then apply;

The subroutine call is:

CALL Poper('name',parameter1,parameter2,...,'apply', ErrHnd)

where:

oper is an abbreviated form of the PS 300 command such as rotate in x -- Protx

'name' is a character string containing the name to be associated with the operate

parameter1,parameter2,..., are the parameters to be used in computing the operation. These may be logicals, integers, reals, vectors, or matrices.

'apply' is a character string containing the name of the object to which this operate applies.

ErrHnd is the user-defined error-handler subroutine.

Example 2

For commands to "send" to functions or display structures, such as

Send datum to <input>dest;

The subroutine call is:

CALL PSNtyp(datum,input,'dest', ErrHnd)

where:

'typ' is an abbreviated form of the PS 300 command such as PSNFIX, PSNM2D,...

datum is what is to be sent. It may be logical, integer, real, character string, vector, or a REAL*4 two dimensional array.

input is an integer which specifies which input of the destination is being sent to.

'dest' is a character string containing the name of the display structure or function.

ErrHnd is the user-defined error-handler subroutine.

Note that the function names in the GSRs are specified without the "F:" prefix that is used in the standard PS 300 command language.

Example 3

For commands which create functions and connections such as:

```
Name := f:genfcn;
Name := f:genfcn(n);
Conn name<output>:<input>dest;
Disc name<output>:<input>dest;
```

The subroutine calls are:

```
CALL PFN     ( 'name', 'genfcn', ErrHnd )
CALL PFNN    ( 'name', 'genfcn', n, ErrHnd )
CALL PCONN   ( 'name',output,input,'dest', ErrHnd )
CALL PDI     ( 'name',output,input,'dest', ErrHnd )
```

where:

'name' is a character string containing the name associated with the function instance.

'genfcn' is a character string containing the name of the system generic function.

n is an integer specifying the number of input/outputs for this function instance.

output,input are integers specifying the output and input numbers.

'dest' is a character string containing the name of the display structure or function.

ErrHnd is the user-defined error-handler subroutine.

EXCEPTIONS

There are two PS 300 commands that use three subroutines. These are the PS 300 LABEL command and the VECTOR_LIST command. For both these commands, the Graphics Support Routines require three separate calls.

To create, specify and complete a label block, the user must call:

PLaBeg - To create and open a label block

PLaAdd - May be called multiple times to add to a previously opened label block

PLaEnd - To complete the creation of a label block.

Together these three subroutines implement the PS 300 command:

```
Name := LABELS x, y, z, 'string'
                  .
                  .
                  .
        x, y, z, 'string';
```

In the same way, the user must call PVcBeg to begin a vector list, PVcLis to send a piece of a vector list, and PVcEnd to end a vector list.

An example of a call that varies slightly from the PS 300 command is the PBSPL call; the PS 300 BSPLINE command. In the PS 300 command language, some of the parameters are optional. In the subroutine they are all required. This is also the case for the PRBSPL, PPOLY, and PRPOLY subroutines.

The PS 300 syntax allows for instancing multiple display entities and for creating multiple variables. In the PS 300 command language the commands would be:

```
NAME := INSTANCE a,b,c,d;
```

for instancing multiple display entities, and

```
VARIABLE s,y,z,w,t,q;
```

for multiple variables.

To perform the equivalent instancing of multiple display entities or for creating multiple variables, the following Graphics Support Routine subroutines should be used.

For the multiple instance case:

```
CALL PINST('NAME', 'A', ErrHnd)
CALL PINCL('B', 'NAME', ErrHnd)
CALL PINCL('C', 'NAME', ErrHnd)
CALL PINCL('D', 'NAME', ERRHND)
```

For the multiple variable case:

```
CALL PVAR ('S', ERRHND)
CALL PVAR ('Y', ERRHND)
CALL PVAR ('Z', ERRHND)
CALL PVAR ('W', ERRHND)
CALL PVAR ('T', ERRHND)
CALL PVAR ('Q', ERRHND)
```

EXCLUDED COMMANDS

There are several classes of commands that were not implemented in the Graphic Support Routines. These include unit commands, commands that are currently being reworked in the PS 300 Graphics Firmware, commands that duplicate functionality, and commands that report the status or the configuration of the PS 300.

Units are handled exclusively by the Parser, and as such cannot be passed as binary data to the Command Interpreter. Commands that are currently being reworked in the firmware will be added to the Graphics Support Routines at a later date. The command status and system configuration commands have no applications in an interactive program.

A list of the excluded commands and the reason for their exclusion is shown in the following table.

TABLE 1

COMMAND	REASON FOR EXCLUSION
Define Units;	Unit command - does not apply
Report Units;	Unit command - does not apply
Forget Units;	Unit command - does not apply
Initialize Units;	Unit command - does not apply
With Pattern;	Currently being reworked
Begin_Font;	Currently being reworked
End_Font;	Currently being reworked
Xform Vector;	Currently being reworked
Xform Matrix;	Currently being reworked
Store;	Duplicated functionality (use SEND TO)
Look From;	Duplicated functionality (use Look AT)
Command Status	Status command
Setup/Show Interface	System configuration command

Except for the exclusions mentioned above, each PS 300 command corresponds to one or more subroutines in the Graphics Support Routines. Commands not implemented in the GSRs are sendable via the PPUTP subroutine which sends the command to the PS 300 Parser.

ERROR HANDLING

An error handling scheme has been employed to catch errors detected by the Graphics Support Routines. Examples of errors detected by the Graphics Support Routines are:

Prefix not followed by an operate.
Follow not followed by an operate.
Multiple calls to PVcLis for block-normalized vector list data.
Invalid characters in a name.

Command Interpreter errors and warnings are not detected by the Graphics Support Routines. Examples of these errors are:

Destination does not yet exist.
Message rejected by destination.
Connection not made.

Error checking will be performed within the Graphics Support Routines to insure that only valid characters are sent within names, and that subroutines are called in the proper order, in cases where order is required. No attempt has been made to capture errors and/or warnings from the Command Interpreter.

Each subroutine call includes an argument that specifies the user-written error handler. This error-handler is of the form:

Subroutine ERRHND (ercode)

where **ercode** is an integer error code corresponding to one of the errors.

WARNING

It is critical that the user specify the error handler as EXTERNAL in all subroutines that make calls to the Graphics Support Routines. Otherwise, the address of a real variable will be passed as a subroutine address and unpredictable events will occur if the error handler is called.

It is the responsibility of the user to provide an error-handling routine to decide what action should be taken when an error is detected. The Graphics Support Routines do not attempt to terminate execution or log errors.

The name, description, and error code of each detectable error is given in tables in the second section of this manual. A sample error-handler subroutine appears in both example programs in Appendix A of this manual. It is a sophisticated error-handler that may be incorporated by the user into an error-handling scheme, or used as an example of what an error-handler should look like.

EXAMPLES OF THE SUBROUTINES

The following two examples show how the subroutines are described in this manual.

EXAMPLE - 1

PS 300 DEC VAX/VMS FORTRAN-77 GSR PROTX

Name := ROTATE in X

APPLICATION SUBROUTINE AND PARAMETERS

CALL PRotX (Name, Angle, Apply, ErrHnd)

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is a user-defined error-handler subroutine.

PS 300 COMMAND AND SYNTAX

Name := ROTate in X Angle (APPLied to Apply);

To use the PROTX call, instead of sending the ASCII command string:

xrot := ROTate in X 37 applied to object;

the application program would call the X-rotation subroutine:

CALL PRotX ('xrot', 37.0, 'object', ErrHnd)

where 'xrot' is the name of the display structure, 37.0 is the angle of X rotation, 'object' is the display structure to which the X rotation is to be applied, and the ErrHnd is a user-defined subroutine that handles errors detected by the Graphics Support Routines.

The ROTATE IN X example is fairly straight forward, as are the majority of the subroutines.

The description of the PCONN subroutine and its parameters is given in the following example.

EXAMPLE - 2

PS 300 DEC VAX/VMS FORTRAN-77 GSR PCONN

Name := CONNECT

APPLICATION SUBROUTINE AND PARAMETERS

CALL PConn (Source, Out, Inp, Dest, ErrHnd)

where:

Source is a CHARACTER STRING
Out is a INTEGER *4
In is an INTEGER *4
Dest is a CHARACTER STRING
ErrHnd is a user-defined error-handler subroutine.

PS 300 COMMAND AND SYNTAX

CONNECT Source <Out>:<Inp> Dest;

Continuing this example, we connect 'name' to the display structure 'xrot' using PConn as follows.

CALL PConn ('name', 1, 1, 'xrot', ErrHnd)

where output <1> of 'name' is connected to input <1> of the display structure 'xrot'.

The PS 300 command syntax for this same operation is:

CONNECT name<1>:<1>xrot;

PROGRAMMING SUGGESTIONS

The file PROCONST.FOR contains definitions for constants used by the Graphics Support Routines. It is often convenient to think of these constants by name rather than by remembering numbers. Specifically, in the usual PS 300 command syntax, inputs to display structures are often referred to by name such as <append> and <clear> for vector lists and <position> and <step> for character strings. There are also <delete>, <last>, and others. Other useful constants such as values for conditional tests for level of detail, and vector list class are obtainable from PROCONST.FOR. PROCONST.FOR also contains a complete set of error/warning code definitions. These values are given in the error table at the end of this manual and may be referenced by name by the user subroutine if PROCONST.FOR is INCLUDED in the subroutine.

The following is an abbreviated list derived from PROCONST.FOR of the constants which should be most useful to the user.

GSR constant declarations:

Name	Meaning
PIAPP:	<Append> input number.
PIDEL:	<Delete> input number.
PICLR:	<Clear> input number.
PISTEP:	<Step> input number.
PIPOS:	<Position> input number.
PILAST:	<Last> input number.
PISUBS	<Substitute> input number.
PCLES:	"Less" level of detail comparison operator.
PCEQL:	"Equal" level of detail comparison operator.
PCLEQL:	"Less-equal" level of detail comparison operator.
PCGTR:	"Greater" level of detail comparison operator.
PCNEQL:	"Not-equal" level of detail comparison operator.
PCGEQL:	"Greater-equal" level of detail comparison operator.
PVCONN:	Vector list "Connected" class type.
PVDOTS:	Vector List "Dots" class type.
PVITEM:	Vector List "Itemized" class type.
PVSEPA:	Vector List "Separate" class type.

INTEGER*4

```
                PIAPP, PIDEL, PICLR,
&               PISTEP, PIPOS, PILAST, PISUBS, PCLES,
&               PCEQL, PCLEQL, PCGTR, PCNEQL, PCGEQL,
&               PVCONN, PVDOTS, PVITEM, PVSEPA
```

PARAMETER

```
                    PIAPP =  0, PIDEL = -1,
     &              PICLR = -2, PISTEP= -3, PIPOS = -4, PILAST= -5,
     &              PISUBS = -6, PCLES = 0, PCEQL =  1, PCLEQL=  2,
     &              PCGTR =  3, PCNEQL=  4, PCGEQL=  5, PVCONN=  0,
     &              PVDOTS=  1, PVITEM=  2, PVSEPA=  3,)
```

The following example illustrate the use of PROCONST.FOR.

Example 3: **Send to a vector list.**

```
      PROGRAM TEST
      INCLUDE ' PROCONST.FOR '
      LOGICAL*1 PL (100)
      DIMENSION VECS( 4,100 ), AVEC( 3 )
      REAL*4  VECS, AVEC
C
C      Always declare user error handler external
C
      EXTERNAL ERRHND
      .
      .
      .
C
C      Create a vector list named VLIST containing 100 connected vectors
C           PVCONN is defined in PROCONST.FOR
C
      CALL PVCBEG ( 'VLIST', 100, .FALSE., .FALSE., 3, PVCONN, ERRHND )
      CALL PVCLIS ( 100, VECS, PL, ERRHND )
      CALL PVCEND ( ERRHND )
C
C      Send a 3d vector to <append> of vecs.
C           PIAPP is defined in PROCONST.FOR.
C
      CALL PSNV3D ( AVEC, PIAPP, 'VLIST', ERRHND )
C
C      Delete 2 vectors from VLIST.
C      PS 300 command: Send fix(2) to <delete>vlist;
C           PIDEL is defined in PROCONST.FOR.
C
      CALL PSNFIX ( 2, PIDEL, 'VLIST', ERRHND )
      .
      .
      .
      END
```

INDEX TO THE SUBROUTINES

The following list from left to right gives an alphabetical listing of the PS 300 Command Name and the FORTRAN Subroutine Name in this manual where the procedure is listed with its parameters.

PS 300 COMMAND NAME	FORTRAN SUBROUTINE	PAGE
DISCONNECT ALL	PDIALL	53
DISCONNECT OUTPUT	PDIOUT	54
DISPLAY	PDISP	55
ENABLE/DISABLE RASTER VIDEO	PRASVI	104
END	PEND	56
END OPTIMIZE	PENDOP	57
END_STRUCTURE	PENDS	58
ERASE PATTERN FROM	PERAPA	59
ERASE_RASTER_SCREEN	PRASER	100
EYE_BACK	PEYEBK	60
F:FUNCTION NAME	PFN	62
F:FUNCTION NAME (INOUTS)	PFNN	63
FIELD_OF_VIEW	PFOV	67
FOLLOW WITH	PFOLL	64
FORGET	PFORG	66
IF CONDITIONAL_BIT	PIFBIT	68
IF LEVEL_OF_DETAIL	PIFLEV	69
IF PHASE	PIFPHA	70
ILLUMINATION	PILLUM	71
INCLUDE	PINCL	72
INCREMENT LEVEL_OF_DETAIL	PINLOD	77
INITIALIZE	PINIT	73
INITIALIZE CONNECTIONS	PINITC	74
INITIALIZE DISPLAYS	PINITD	75
INITIALIZE NAMES	PINITN	76

PS 300 COMMAND NAME	FORTRAN SUBROUTINE	PAGE
INSTANCE OF	PINST	78
LABLES	PLAADD	79
	PLABEG	80
	PLAEND	81
LOAD PIXEL VALUE	PRASWP	105
LOOK_AT_FROM	PLOOKA	82
MATRIX_2X2	PMAT22	83
MATRIX 3X3	PMAT33	84
MATRIX 4X3	PMAT43	85
MATRIX 4X4	PMAT44	86
NIL	PNIL	87
OPTIMIZE STRUCTURE	POPT	88
PATTERN	PDEFPA	48
PATTERN Name1 WITH Name2	PPATWI	89
POLYGON (ATTRIBUTES)	PPLYGA	90
POLYGON (BEGIN)	PPLYGB	91
POLYGON (END)	PPLYGE	92
POLYGON (LIST)	PPLYGL	93
POLYGON (OUTLINE)	PPLYGO	95
POLYNOMIAL	PPOLY	96
PREFIX NAME WITH	PPREF	98
RATIONAL BSPLINE	PRBSPL	107
RATIONAL POLYNOMIAL	PRPOLY	116
RAWBLOCK	PRAWBL	106
REMOVE FOLLOWER OF NAME	PREMFO	110
REMOVE FROM	PREMFR	111

PS 300 COMMAND NAME	FORTRAN SUBROUTINE	PAGE
REMOVE NAME	PREM	109
REMOVE PREFIX	PREMPR	112
RESERVE_WORKING_STORAGE	PRSVST	118
ROTATE IN X	PROTX	113
ROTATE IN Y	PROTY	114
ROTATE IN Z	PROTZ	115
SCALE	PSCALE	119
SECTIONING_PLANE	PSECPL	127
SEND (RAW) STRING TO	PSNRST	148
SEND (VECTOR LIST)	PSNVL	154
SEND 2D MATRIX TO	PSNM2D	143
SEND 2D VECTOR TO	PSNV2D	150
SEND 3D MATRIX TO	PSNM3D	144
SEND 3D VECTOR TO	PSNV3D	151
SEND 4D MATRIX TO	PSNM4D	145
SEND 4D VECTOR TO	PSNV4D	152
SEND BOOLEAN TO	PSNBOO	141
SEND COUNT*DRAWMV TO	PSNPL	146
SEND FIX TO	PSNFIX	142
SEND REAL_NUMBER TO	PSNREA	147
SEND STRING TO	PSNST	149
SEND VALUE TO	PSNVAL	153
SET CHARACTERS SCREEN_ORIENTED	PSECHS	122
SET CHARACTERS SCREEN_ORIENTED/FIXED	PSECHF	121
SET CHARACTERS WORLD_ORIENTED	PSECHW	123

PS 300 COMMAND NAME	FORTRAN SUBROUTINE	PAGE
SET COLOR	PSECOL	125
SET COLOR BLENDING	PSETCB	140
SET CONDITIONAL_BIT	PSEBIT	120
SET CONTRAST	PSECON	126
SET CSM	PSECSM	128
SET DEPTH_CLIPPING	PSEDCL	130
SET DISPLAY	PSEDOF	131
SET DISPLAYS ALL	PSEDAL	129
SET INTENSITY	PSEINT	132
SET LEVEL_OF_DETAIL	PSELOD	133
SET LOGICAL DEVICE COORDINATES	PRASLD	101
SET LOOK_UP_TABLE_RANGE	PRASLR	102
SET PICKING INDENTIFIER	PSEPID	134
SET PICKING LOCATION	PSEPLO	135
SET PICKING OFF	PSEPOF	137
SET PIXEL LOCATION	PRASCP	99
SET PLOTTER	PSEPLT	136
SET RATE	PSER	138
SET RATE EXTERNAL	PSEREX	139
SETUP CNESS	PSECNS	124
SOLID_RENDERING	PSOLRE	155
STANDARD FONT	PSTDFO	156
SURFACE_RENDERING	PSURRE	157
TRANSLATE	PTRANS	158

PS 300 COMMAND NAME	FORTRAN SUBROUTINE	PAGE
VARIABLE NAME	PVAR	159
VECTOR_LIST	PVCBEG	160
	PVCEND	162
	PVCLIS	163
VIEWPORT	PVIEWP	165
WINDOW	PWINDO	166
WRITE LOOK_UP_TABLE ENTRIES	PRASLU	103
XFORM MATRIX	PXFMAT	168
XFROM VECTOR_LIST	PXFVEC	169

The following is a list of the Utility Subroutines.

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 three interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
DMR-11 (high-speed synchronous interface)
PARALLEL (high speed parallel interface

The parameter string must contain EXACTLY 1 "/" somewhere between the above phrases. 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)

(continued)

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 SUBROUTINE AND PARAMETERS

```
CALL PDelim (Newd, ErrHnd)
```

where:

Newd is a single character CHARACTER STRING that is the new string delimiter

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine can be used to change the string delimiting character. The default string delimiter is " (double quote).

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PDtach (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine detaches (disconnects) the communications link established between the host and the PS 300.

PDtach should always be the last GSR subroutine called by the user application program.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PGet (Str, MsgLen, ErrHnd)
```

where:

Str is a CHARACTER STRING that contains the message read from the PS 300

MsgLen is an INTEGER*4 that returns the number of bytes read from the PS 300

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

The PGet subroutine is used to poll the PS 300 for input records by requesting a message that has been sent to the PS 300 function HOST_MESSAGE. The actual message contents are returned in: Str. The number of bytes read are returned in Msglen.

If a PGet call is issued and no message exists to be sent back to the host, then the returned length of the message (Msglen) is 0. Otherwise, the length of the message is greater than 0, and indicates the true number of bytes in the message.

NOTE

If the default value for input <2> or input <3> of HOST_MESSAGEB is changed by the user to be something other than a single carriage return, then the above description no longer applies. The user should refer to Appendix B of this manual for a description of HOST_MESSAGEB and its inputs.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PGetW (Str, MsgLen, ErrHnd)
```

where:

Str is a CHARACTER STRING that contains the message read from the PS 300

MsgLen is an INTEGER*4 that returns the number of bytes read from the PS 300

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

The PGetW subroutine is used to query the PS 300 for input records by requesting a message that has been sent to the PS 300 function HOST_MESSAGE. If no message exists to be read, the PGetW subroutine will wait until a message arrives from HOST_MESSAGE. The actual message contents are returned in: Str. The number of bytes read are returned in Msglen.

NOTE

If the default value for input <2> or input <3> of HOST_MESSAGEB is changed by the user to be something other than a single carriage return, then the above description no longer applies. The user should refer to Appendix B of this manual for a description of HOST_MESSAGEB and its inputs.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PmuxCI (CIchan, ErrHnd)
```

where:

CIchan is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine redefines the CIROUTE output channel accessed as the Binary CI channel. The standard and default CI channel is 2.

This subroutine is provided to allow for the implementation of multiple command interpreters.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PMuxG (MuxChn, ErrHnd)
```

where:

Muxchn is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine redefines the CIROUTE output channel being currently accessed as the "generic" channel by PPutG. The call is provided to support the future implementation of custom user-functions connected to various outputs of CIROUTE.

MuxChn = 1: Send to parser. CIROUTE<3>
MuxChn = 2: Send to READSTREAM CIROUTE<4>
etc.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PMuxP (PrsChn, ErrHnd)
```

where:

PrsChn is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine redefines the CIROUTE output channel accessed by PPutP. The call allows for the implementation and support of multiple Parsers. The standard and default Parser channel is 1.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PPurge (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

The GSRs always buffer the output to the PS 300. This subroutine insures that the output buffer is flushed.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PPutG (String, Length, ErrHnd)
```

where:

String is a CHARACTER STRING
Length is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends the bytes specified in the buffer: String to the current generic demultiplexing channel of CIROUTE established by: PMuxG. Length defines the number of bytes to send.

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PPutP (String, Length, ErrHnd)
```

where:

String is a CHARACTER STRING
Length is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends the characters specified in the buffer: String to the PS 300 parser. Length defines the number of bytes to send.

ALLOCATE PLOTTER

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PALLPL (Plot, ErrHnd)
```

where

Plot is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine allocates the plotter specified in PLOT to the calling user. When the plotter is allocated, formfeed after plot is disabled.

PS 300 COMMAND AND SYNTAX

ALLOCATE PLOTTER Plot;

Name := ATTRIBUTES

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR (Name, Hue, Sat, Intens, Reserv, Diffus, Specul, ErrHnd)
```

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Reserv is a REAL
Diffus is a REAL
Specul is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. Hue, Sat and Intens define the color of the polygon. Hue specifies an angle between 0 and 360 indicating the color on a color wheel with full blue begin 0, red being 120 and green being 240. Sat specifies the saturation of the color with 0 being no color and 1 being full saturation. Intens specifies the intensity of the color with 0 being no color (black) and 1 being full intenstiy. Diffus is the proportion of color contributed by defuse reflection versus that contributed by specular reflection with a value of 1 eliminating all specular highlighting and a value of 0 eliminating all diffuse reflectivity. Specul adjusts the concentration of specular highlights in the range of 0 to 10.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR Hue[,Sat[Intens]]]
                   [DIFFUSE Diffus]
                   [SPECULAR Specul];
```

Name := ATTRIBUTES ... AND

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR2 (Name, Hue, Sat, Intens, Reserv, Diffus, Specul,
             Hue2, Sat2, Inten2, Reser2, Diffu2, Specul2, ErrHnd)
```

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Reserv is a REAL
Diffus is a REAL
Specul is an INTEGER*4
Hue2 is a REAL
Sat2 is a REAL
Inten2 is a REAL
Reser2 is a REAL
Diffu2 is a REAL
Specul2 is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. This is similar to the PATTR subroutine but allows for a second set of attributes to be defined for the backside of polygons.

PS 300 COMMAND AND SYNTAX

```
Name :=  ATTRIBUTES  [COLOR Hue[,Sat[Intens]]]
                     [DIFFUSE Diffus]
                     [SPECULAR Specul];
         AND         [COLOR Hue2[,Sat2[,Inten2]]]
                     [DIFFUSE Diffu2]
                     [SPECULAR Specu2];
```

Name := BEGIN

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PBeg (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This call is used with the PEND subroutine to group a set of viewing and/or modeling commands so that they appear to be executed simultaneously.

PS 300 COMMAND AND SYNTAX

Name := BEGIN

Name := BEGIN_S

APPLICATION SUBROUTINE AND PARAMETERS

```
CAll PBegs (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine is used with the PENDS subroutine to group a set of viewing and/or modeling commands so that each element does not need to be explicitly named to be accessed.

PS 300 COMMAND AND SYNTAX

Name := BEGIN_Structure

Name := BSPLINE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PBspl (Name, Order, OpnCls, NonPer, Dimen, NVert, Vertic,
            KntCnt, Knots, Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING specifying the name to be assigned to the computed B-spline

Order is an INTEGER*4 specifying the order of the B-spline

For OpnCls .TRUE. is Open and .FALSE. is Closed

For NonPer .TRUE. is Non/periodic and .FALSE. is Periodic

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

NVert is an INTEGER*4 specifying the number of vertices

Vertic is defined: REAL*4 Vertic (4, NVert) specifying the vertices
where: Vertic (1,n) = x (n)
Vertic (2,n) = y (n)
Vertic (3,n) = z (n)
Vertic (4,n) is not used.

KntCnt is an INTEGER*4 specifying the number of knots

Knots is an array (KntCnt +1) of REAL *4 specifying the knot sequence

Chords is an INTEGER*4 specifying the number of vectors to be created

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine evaluates a B-spline curve, allowing the parametric description of the curve form without having to specify the coordinates of each vector.

(Continued on next page)

(continued)

PS 300 COMMAND AND SYNTAX

```
Name := BSPLINE
        ORDER = Order
        OPEN/CLOSED
        NONPERIODIC/PERIODIC
        N = NVert
        VERTICES     = X(1),  Y(1),  ( Z(1) )
                       X(2),  Y(2),  ( Z(2) )
                        :     :       :
                       X(N),  Y(N),  ( Z(N) )
        KNOTS = Knots (1), ... Knots (KntCnt)
        CHORDS = Chords;
```

NOTE

None of the parameters in the application subroutine PBSPL are optional. The dimension must be specified in the PBSPL application subroutine. In the PS 300 command, dimension is implied by syntax.

If KntCnt = 0, then the default knot sequence is generated and the knot array is ignored.

Name := CHARACTER ROTATE

APPLICATION SUBROUTINE AND PARAMETERS

```
Call PChRot (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine rotates the specified characters (Apply) and has the following parametric definition:

- Angle is the Z-rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := CHARacter ROTate Angle (APPLied to Apply);

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PChs (Name, TranX, TranY, TranZ, StepX, StepY, Chars, ErrHnd)
```

where:

Name is a CHARACTER STRING
TránX, TranY, TranZ are REAL*4
StepX, StepY are REAL*4
Chars is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine defines a character string (Chars) and specifies its location and placement. It has the following parametric definitions:

- TranX, TranY, TranZ give the x,y,z coordinates of the location of the beginning of the character string
- StepX, StepY give the spacing between characters in character unit size

PS 300 COMMAND AND SYNTAX

```
Name := CHARacters TranX,TranY,TranZ STEP StepX,StepY 'Chars';
```

Name := CHARACTER SCALE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PChSca (Name, ScaleX, ScaleY, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
ScaleX, ScaleY are REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a uniform 2x2 scale matrix to scale the specified characters (Apply). It has the following parametric definition:

- ScaleX, ScaleY give the scaling factors for the x,y axes

PS 300 COMMAND AND SYNTAX

Name := CHARacter SCALE ScaleX, ScaleY (APPLied to Apply);

Name := CONNECT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PConn (Source, Out, Inp, Dest, ErrHnd)
```

where:

Source is a CHARACTER STRING
Out is a INTEGER*4
Inp is an INTEGER*4
Dest is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine connects the output (Out) of the function instance (Source) to the input (Inp) of the function instance or display data structure (Dest).

PS 300 COMMAND AND SYNTAX

CONNECT Source <Out>:<Inp> Dest;

Name := COPY

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PCopyV (Name, CpyFrm, Start, Count, ErrHnd)
```

where:

Name is a CHARACTER STRING
CpyFrm is a CHARACTER STRING
Start is an INTEGER*4
Count is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a vector list (Name) containing a group of consecutive vectors copied from another vector list (CpyFrm) where 'Start' is the first vector to be copied and 'Count' is the number of vectors to be copied.

PS 300 COMMAND AND SYNTAX

Name := COPY CpyFrm (START=) Start (,) (COUNT=) Count;

DEALLOCATE PLOTTER

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDALLP (Plot, ErrHnd)
```

where

Plot is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine deallocates a plotter previously allocated to the calling user.

PS 300 COMMAND AND SYNTAX

```
DEALLOCATE PLOTTER Plot;
```

Name := PATTERN

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDEFPA (Name, Segs, Pattrn, Contin, Match, Length, ErrHnd)
```

where

Name is a CHARACTER STRING
Segs is an INTEGER*4
Pattrn is an INTEGER*4 (Segs) Array
Contin is a LOGICAL
Match is a LOGICAL
Length is a REAL
ERRhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines a pattern that can be used to pattern a vector list or curve. Segs defines the number of integers used to define the pattern, those integers given by pattrn. Contin tells whether or not patterning is to go across multiple vectors. Match tells if the pattern length is to be adjusted to make the patterning terminate precisely at the endpoints. Length gives the pattern length.

PS 300 COMMAND AND SYNTAX

```
Name := PATTERN Pattrn [Pattrn(2)...Pattrn(Segs)]
                [AROUND_CORNERS]   [MATCH/NOMATCH]   LENGTH
                Length;
```

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDelet (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine deletes the previously defined data structure name. After a PDelet call is issued, all functions and data structures referring to (Name) will no longer include the data that was associated with (Name).

PS 300 COMMAND AND SYNTAX

DELete Name;

Name := DECREMENT LEVEL_OF_DETAIL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDeLOD (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine decrements the current level of detail by 1.

PS 300 COMMAND AND SYNTAX

Name := DECrement LEVel_of_detail (APPLied to Apply);

DEL NAME*

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDELW (Name, ErrHnd)
```

where

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

DESCRIPTION

This subroutine deletes all names that begin with the characters specified in the parameter name.

PS 300 COMMAND AND SYNTAX

DELETE Name*;

DISCONNECT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDi (Source, Out, Inp, Dest, ErrHnd)
```

where:

Source is a CHARACTER STRING
Out is an INTEGER*4
Inp is an INTEGER*4
Dest is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine disconnects the output (Out) of the function instance (Source) from the input (Inp) of the function instance or display data structure (Dest).

PS 300 COMMAND AND SYNTAX

DISCONNect Source <Out>:<Inp> Dest;

DISCONNECT Source:ALL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDiAll (Source, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine disconnects all outputs of (Source) from all inputs to function instances or display data structures that it was previously connected to.

PS 300 COMMAND AND SYNTAX

DISCONNect Source:ALL;

DISCONNECT <OUT>

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDiout (Source, Out, ErrHnd)
```

where:

Source is a CHARACTER STRING
Out is an INTEGER*4
Errhnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine disconnects the output (Out) of the function instance (Source) from all inputs to function instances or display data structures it was previously connected to.

PS 300 COMMAND AND SYNTAX

DISCONNect Source <Out>:ALL;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PDisp (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine displays a data structure (Name).

PS 300 COMMAND AND SYNTAX

DISPlay Name;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PEnd (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine is used with the PBEGIN subroutine to group a set of viewing and/or modeling commands so that they appear to be executed simultaneously.

PS 300 COMMAND AND SYNTAX

```
END;
```

END OPTIMIZE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PEndOp (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine is used with the POptSt subroutine. When POptSt is called, it places the PS 300 in an "optimization mode" in which certain elements of the display data structure are created in a way that minimizes Display Processor traversal time. PEndOp must be called to complete the sequence.

It is strongly suggested that users familiarize themselves with the OPTIMIZE command documentation in the **PS 300 Command Summary** before using this subroutine to learn the full ramifications and constraints of this command.

PS 300 COMMAND AND SYNTAX

```
END OPTIMIZE;
```

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PEnds (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine is used with the PBEGS subroutine to group a set of viewing and/or modeling commands so that each element does not need to be explicitly named to be accessed.

PS 300 COMMAND AND SYNTAX

END_Structure;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PERAPA (Name, ErrHnd)
```

where

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

DESCRIPTION

This subroutine removes a pattern from name if name is a patterned vector list or curve.

PS 300 COMMAND AND SYNTAX

ERASE PATTERN FROM Name;

Name := EYE BACK

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PEyeBk (Name, DBack, DistLR, DistUD, Wide, Front, Back,
             Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
DBack is a REAL*4
DistLR is a REAL*4 (positive for right/negative for left)
DistUD is a REAL*4 (positive for up/negative for down)
Wide is a REAL*4
Front is a REAL*4
Back is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies a viewing pyramid with the following parametric definitions:

- DBack is the perpendicular distance of the eye from the plane of the viewport
- DistHoriz is the distance of the eye right or left from the viewport center (positive for right/negative for left)
- DistVert is the distance from the eye up or down from the viewport center (positive for up/negative for down)
- Wide is the width of the viewport
- Front is the front boundary of the frustum of the viewing pyramid
- Back is the back boundary of the frustum of the viewing pyramid

(Continued on next page)

Name := EYE BACK

(continued)

PS 300 COMMAND AND SYNTAX

Name := EYE BACK distb
[left]/[right] distlr
[up]/[down] disud
from screen area wide
Front Boundary = front
Back Boundary = back
(APPLIED to Apply);

NOTE

PS 300 syntax allows specification of both left and right and up and down in the same command, which results in an accumulation of right/left and up/down. PEYEBK allows only signed real numbers that if positive specify right and up, and if negative specify left and down.

Example:

```
eye_spec:=  eye back .6 left 2.5 right 3 up 2.1 down 6 from screen area 2
            wide front=.0001 back=100 then apply;
```

is equivalent to:

```
eye_spec:=  eye back .6 right .5 down -3.9 from screen area 2 wide
            front=.0001 back=100 then apply;
```

and has the same effect as:

```
CALL PEYEBK ('EYE_SPEC', 0.6,0.5,-3.9,2.0,.0001,100.0,'APPLY',ERRHND)
```

Name := F:FUNCTION NAME

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PFn (Name, FnName, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine creates an instance of an intrinsic PS 300 function.

PS 300 COMMAND AND SYNTAX

Name := F:FnName;

Name := F:FUNCTION NAME (INOUTS)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PFnN (Name, FnName, InOuts, ErrHnd)
```

where:

Name is a CHARACTER STRING
FnName is a CHARACTER STRING
InOuts is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates an instance of an intrinsic PS 300 function where InOuts is the number of respective inputs or outputs of the function. Intrinsic functions that are used by this subroutine are F:Route(n), F:RouteC(n), F:Inputs_Choose(n), and F:SYNC(n).

PS 300 COMMAND AND SYNTAX

Name := F:FnName (InOuts);

FOLLOW WITH

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PFoll (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine follows a named operation node (Name) with another operation node. To use the PFOLL subroutine, the user must first call this subroutine and then the user MUST IMMEDIATELY call the subroutine corresponding to the "transformation-or-attribute command".

PS 300 COMMAND AND SYNTAX

FOLLOW name WITH transformation-or-attribute command;

Example:

PS 300 Command

FOLLOW xrot WITH scale by .5,.5,.5;

would be:

```
REAL*4 V(3)
CHARACTERS*1 Null
DATA V,NULL /0.5,0.5,0.5,""/
  .
  .
  .
  .
CALL PFOLL ('xrot', ErrHnd)
CALL PSCALE (Null, V, Null, ErrHnd)
```

Name := CHARACTER FONT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PFont (Name, FontNm, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
FontNm is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine establishes a character font (FontNm) as the working font for the specified display structure (Apply).

PS 300 COMMAND AND SYNTAX

Name := CHARACTER FONT FontNm (APPLied to Apply);

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PForg (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine removes (Name) from the display and from the name directory, where (Name) is any previously defined data structure name.

PS 300 COMMAND AND SYNTAX

FORget Name;

Name := FIELD_OF_VIEW

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PFov (Name, Angle, Front, Back, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Front is a REAL*4
Back is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies a right rectangular viewing pyramid with the following parametric definitions:

- Angle is the angle of view from the eye
- Front is the front boundary of the frustum of the viewing pyramid
- Back is the back boundary of the frustum of the viewing pyramid

PS 300 COMMAND AND SYNTAX

```
Name := Field_Of_View Angle
        FRONT boundary = Front
        BACK boundary  = Back
        (APPLied to Apply);
```

Name := IF CONDITIONAL_BIT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PIfBit (Name, BitNum, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
BitNum is an INTEGER*4
OnOff is .TRUE. for ON, .FALSE. for OFF
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine refers to a data structure if an attribute bit has a specified setting (On or Off), with the following parametric definitions:

- Bit Number indicates which bit to test

PS 300 COMMAND AND SYNTAX

Name := IF conditional_BIT BitNum is OnOff (THEN Apply);

Name := IF LEVEL_OF DETAIL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PIfLev (Name, Level, Comp, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Level is an INTEGER*4
*Comp an INTEGER*4 corresponding to the comparison test to be performed.
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine refers to a data structure if the level of detail attribute has a specified relationship to a given number, with the following parametric definitions:

- Level indicates the number to compare with the current level of detail
- *Comparison corresponds to the comparison test to be performed.

PS 300 COMMAND AND SYNTAX

Name := IF LEVEL_of_detail Comp Level (THEN Apply);

* 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 short table of the mnemonics and their INTEGER*4 value is given below.

Mnemonic	Comparision	INTEGER*4 Value
PCLES	<	0
PCEQL	=	1
PCLEQL	<=	2
PCGTR	>	3
PCNEQL	<>	4
PCGEQL	>=	5

Name := IF PHASE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PIfPha (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Onoff is .TRUE. for On and .FALSE. for Off
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine refers to a data structure if the PHASE attribute is in the specified state, ON or OFF.

PS 300 COMMAND AND SYNTAX

Name := IF PHASE OnOff (THEN Apply);

Name := ILLUMINATION

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PILLUM (Name, X, Y, Z, Hue, Sat, Intens, Ambien, ErrHnd)
```

where

Name is a CHARACTER STRING
X is a REAL
Y is a REAL
Z is a REAL
Hue is a REAL
Sat is a REAL
Intens is a REAL
Ambien is a REAL
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines polygon illumination characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. The direction to the light source is specified by x,y, z. The color is specified by Hue, Sat and Intens. Its contribution to ambient lighting is specified by Ambien (0 to 1).

PS 300 COMMAND AND SYNTAX

```
Name := ILLUMINATION X, Y, Z
                     [COLOR Hue[,Sat[,Intens]]]
                     [AMBIENT Ambien];
```

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PIncl (Name1, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING
Name2 is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine is used to include one named display data structure (Name1) in a named instance of another display data structure (Name2).

PS 300 COMMAND AND SYNTAX

INCLude Name1 IN Name2;

INITIALIZE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PInit (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine restores the PS 300 to its initial state; there are no user-defined names, display data structures, or function connections, and no data structures are displayed.

PS 300 COMMAND AND SYNTAX

INITialize;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PInitC (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine breaks all user-defined function connections.

PS 300 COMMAND AND SYNTAX

INITialize CONNections;

INITIALIZE DISPLAYS

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PInitD (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine removes all display data structures from the display list.

PS 300 COMMAND AND SYNTAX

INITialize DISPlays;

INITIALIZE NAMES

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PInitN (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine clears the name dictionary of all display data structures and function instance names.

PS 300 COMMAND AND SYNTAX

INITialize NAMES;

Name := INCREMENT LEVEL_OF_DETAIL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PInLOD (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine increments the current level of detail by 1.

PS 300 COMMAND AND SYNTAX

Name := INCRement LEVel_of_detail (APPLied to Apply);

Name1:= INSTANCE OF

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PInst (Name1, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING
Name2 is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates an instance node (Name1) with pointers to the data structure referenced (Name2).

PS 300 COMMAND AND SYNTAX

Name1 := INSTance (of Name2);

Name := LABELS (no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PLaAdd (X, Y, Z, Label, ErrHnd)
```

where:

X,Y,Z are REAL*4
Label is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine is the middle call in creating a label block. It must be called to add a label to a previously opened label block created by the call to: PLaBeg. To create a label block, the user must call:

PLaBeg
PLaAdd (This call may be made multiple times)
PLaEnd

PS 300 COMMAND AND SYNTAX

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

```
Name := LABELS  x, y, z, 'string'
                  :
                  :
                x, y, z, 'string';
```

Name := LABELS (no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PLaBeg (LabBlk, StepX, StepY, ErrHnd)
```

where:

LabBlk is a CHARACTER STRING
StepX, StepY are REAL*4
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to create and open a label block. To complete the label block call and specify a label block, the user must call:

PLaBeg
PLaAdd (This call may be made multiple times)
PLaEnd

PS 300 COMMAND AND SYNTAX

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

```
Name := LABELS  x, y, z, 'string'
                   :
                   :
                x, y, z, 'string';
```

NOTE

The stepx and stepy parameters allow the steps between the label blocks to be specified in terms of x and y. If stepx and stepy were specified as 1.0 and 0.0 respectively, each successive character would be displayed one unit to the right of and horizontally aligned with the preceding character. This applies to all labels within the label block. It should prove useful for those users who wish to make vertical or slanted label blocks. Users cannot send to ‹step› of a label block; a message from the CI results.

Name := LABELS (no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PLaEnd (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to complete the creation of a label block. To completely specify a label block, the user must call:

PLaBeg,
PLaAdd (This call may be made multiple times), and lastly,
PLaEnd.

PS 300 COMMAND AND SYNTAX

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

```
Name := LABELS  x, y, z, 'string'
                  :
                  :
                x, y, z, 'string';
```

Name := LOOK_AT_FROM

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PLookA (Name, At, From, Up, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
At is defined: REAL*4 AT (3)
From is defined: REAL*4 From (3)
Up is defined: REAL*4 Up (3)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine, when used with PWINDO, PEYEBCK, or PFOV, fully specifies the portion of the data space that will be viewed as well as the viewer's orientation in data space. It has the following parametric definitions:

- At is the point being looked at in data space coordinates
- From is the location of the viewer's eye in data space coordinates
- Up indicates the screen "up" direction

PS 300 COMMAND AND SYNTAX

Name := LOOK AT At FROM From UP Up (APPLied to Apply);

Name := MATRIX_2x2

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PMat22 (Name, Mat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a special 2x2 transformation matrix that applies to the specified data (vector lists and/or characters) that follow (Apply).

PS 300 COMMAND AND SYNTAX

Name := Matrix_2x2 Mat (APPLied to Apply);

Name := MATRIX_3x3

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PMat33 (Name, Mat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a special 3x3 transformation matrix that applies to the specified data (vector lists and/or characters) that follow (Apply).

PS 300 COMMAND AND SYNTAX

Name := Matrix_3x3 Mat (APPLied to Apply);

Name := MATRIX_4x3

APPLICATION SUBROUTINE AND PARAMETERS

CALL PMat43 (Name, Mat, Vec, Apply, ErrHnd)

where:

Name is a CHARACTER STRING

Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)

Vector is the x,y,z translation to be sent and is defined: REAL*4 Vec (3)

Apply is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a special 4x3 matrix that applies to the specified data (vector lists and/or characters) that follow (Apply).

PS 300 COMMAND AND SYNTAX

Name := Matrix_4x3 Mat Vec (APPLied to Apply);

NOTE

The matrix_4x3 command is sent in two parts:

1) a 3x3 matrix sent in Mat
2) a 3d vector (4th row) sent in Vec

Name := MATRIX_4x4

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PMat44 (Name, Mat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a special 4x4 matrix that applies to the specified data (vector lists and/or characters) that follow (Apply).

PS 300 COMMAND AND SYNTAX

Name := Matrix_4x4 Mat (APPLied to Apply);

Name := NIL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PNIL (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine names a null data structure. When this subroutine is used to redefine (Name), (Name) is kept in the name directory but any definition previously associated with (Name) is removed. PForget does just the opposite of PNil.

PS 300 COMMAND AND SYNTAX

Name := NIL;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL POpt (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine is used with the PEndOp subroutine. When POpt is called, it places the PS 300 in an "optimization mode" in which certain elements of the display data structure are created in a way that minimizes Display Processor traversal time. PEndOp must be called to complete the sequence.

It is strongly suggested that users familiarize themselves with the OPTIMIZE command documentation in the **PS 300 Command Summary** before using this subroutine to learn the full ramifications and constraints of this command.

PS 300 COMMAND AND SYNTAX

```
OPTIMIZE STRUCTURE;
```

PATTERN Name1 WITH Name2

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPATWI (Name, Patnam, ErrHnd)
```

where

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

DESCRIPTION

This subroutine patterns the curve of the vector_list called Name with the pattern Patnam, where Patnam has been defined with a call to the subroutine PDEFPA.

PS 300 COMMAND AND SYNTAX

PATTERN Name WITH Patnam;

Name := POLYGON (ATTRIBUTES - no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGA (Attr, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine specifies that the attributes named by Attr and specified in a call to PATTR or PATTR2 apply to all subsequent polygons until superceded by another call to PPLYGA.

This subroutine is one of five subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :        :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYGON (BEGIN - no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPlygB (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine begins a polygon display list. The parameter (Name) specifies the name to be given to the polygon display list defined by calls to PPLYGA, PPLYGO and PPLYGL.

This subroutine is one of five subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :        :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

A sequence of 3 to 5 subroutines must be called to create a polygon display vector list:

PPLYGB: This subroutine is called to begin the creation of a polygon vector list.

PPLYGA: This is an optional subroutine called to specify the attribute to be applied to the polygon.

PPLYGO: This is an optional subroutine called to specify the intensity or color of the polygon on the calligraphic display.

PPLYGL: This subroutine specifies the vectors of each polygon in the polygon display list.

PPLYGE: This subroutine closes the polygon display list.

Name := POLYGON (END - no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGE (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine ends the definition of a polygon display list.

This subroutine is one of five subroutines required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :        :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYGON (LIST - no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGL (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ErrHnd)
```

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar

- NVert specifies the number of vertices in the polygon

- Vertic specifies the vertices of the polygon
  Vertic (1, n) = vertex n: x-coordinate;
  Vertic (2, n) = vertex n: y-coordinate;
  Vertic (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Vertic.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

Name := POLYGON (LIST - no corresponding command)

(continued)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.
  This parameter is presently ignored and reserved for future use.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Vertic. This parameter is reserved for future use.

This subroutine is one of five subroutines required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
           :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYGON (OUTLINE - no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGO (Outlin, ErrHnd)
```

where

Outlin is a REAL
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine specifies that Outln be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges on the calligraphic display until superceded by another call to PPLYGO.

This subroutine is one of five subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )
          :        :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] ( [S] x,y,z [N x,y,z] )];
```

Name := POLYNOMIAL

APPLICATION SUBROUTINE AND PARAMETERS

CALL PPoly (Name, Order, Dimen, Coeffs, Chords, ErrHnd)

where:

Name is a CHARACTER STRING

Order is an INTEGER*4

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

Coeffs is defined: REAL*4 Coeffs (4,Order+1)
where: Coeffs (1,i) = x(order -i+1)
Coeffs (2,i) = y(order -i+1)
Coeffs (3,i) = z(order -i+1)
Coeffs (4,i) is not used

:
:
etc.

To further clarify the description:

Coeffs(1,1) = the coefficient that will be applied to the t^{order} term

Coeffs(1,2) = the coefficient that will be applied to the $t^{order-1}$ term in the resultant x(t) function computed by this command.

Chords is an INTEGER*4

ErrHnd is the user-defined error-handler subroutine.

(Continued on next page)

Name := POLYNOMIAL

(continued)

DESCRIPTION

This subroutine allows the parametric description of many curve forms without the need to specify or transfer the coordinates of each constituent vector. It has the following parametric definitions:

- Order is the order of the polynomial
- Coefficients represent the x,y,z components of the curve
- Chords is the number of vectors to be created

PS 300 COMMAND AND SYNTAX

```
Name := POLYNOMIAL
        ORDER = Order
        COEFFICIENTS = X(i),   Y(i),   Z(i)
                       X(i-1), Y(i-1), Z(i-1)
                         :       :       :
                       X(0),   Y(0),   Z(0)
        CHORDS = Chords;
```

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPref (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine prefixes a named display structure (Name) with an operation node. To prefix something, the user must first call this subroutine and then the user MUST IMMEDIATELY call the subroutine corresponding to the: "transformation-or-attribute" command.

PS 300 COMMAND AND SYNTAX

PREfix Name WITH transformation-or-attribute command;

Example:

PS 300 Command:

```
Prefix xrot with translate by .5,.5,0;
```

would be:

```
REAL*4  V(3)
CHARACTER *1 NULL
DATA V,NULL, /0.5,0.5,0.0,""/
  .
  .
  .
  .
CALL PREF ('xrot', ErrHnd)
CALL PTRANS (NULL, V, NULL, ErrHnd)
```

SET PIXEL LOCATION - RASTER ROUTINE

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASCP (x, y, ErrHnd)
```

where:

x is an INTEGER*4
y is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine

DEFINITION

This subroutine establishes the current pixel location relative to the current logical device coordinates. (x) and (y) specify the x,y coordinates of the current pixel and must be greater than or equal to 0.

(0,0) is the lower-left corner of the logical device coordinates.

ERASE_RASTER_SCREEN - RASTER ROUTINE

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASER (Color, ErrHnd)
```

where:

Color is an INTEGER*4 (3)
ErrHnd is the user-defined error-handler subroutine

DEFINITION

This subroutine is used in WRPIX mode to erase the entire screen to the color specified in the parameter (Color), where:

Color(1) is the red index
Color(2) is the green index
Color(3) is the blue index

The index refers to the color table that contains the actual value used for display.

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASLD (Xmin, Ymin, Xmax, Ymax, ErrHnd)
```

where:

Xmin is an INTEGER*4
Ymin is an INTEGER*4
Xmax is an INTEGER*4
Ymax is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine

DEFINITION

This subroutine sets the logical device coordinates that are used to position the picture in virtual address space. The Raster Option has a virtual pixel address space from -32768 to 2047 in both x and y. The portion of this space that is actually displayed is from 0 to 639 in x and from 0 to 479 in y. This subroutine can be used to reposition an image in screen space without re-calculation and only retransmission of the data.

SET LOOK_UP_TABLE_RANGE - RASTER ROUTINE

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASLR (Min, Max, ErrHnd)
```

where:

Min is an INTEGER*4
Max is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine is used in WRLUT mode to to set the Look-Up Table range. This subroutine set the limits within which the LUT entries can be changed. (Min) and (Max) set the minimum and maximum range of the Look-up tables: they must be greater than or equal to 0 and less than 256.

WRITE LOOK_UP_TABLE ENTRIES - RASTER ROUTINE

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASLU (Num, Index, Lutval, ErrHnd)
```

where:

Num is an INTEGER*4
Index is an INTEGER*4
Lutval is an INTEGER*4 (Num,4) Array
ErrHnd is the user-defined error-handler subroutine

DEFINITION

This subroutine sets the current Look-Up Table location and loads the Look-Up Tables. (Num) specifies the number entries in the Lutval parameter. (Index) specifies the location in the Look-Up Table where the entries will start being loaded and (Lutval) is the (Num,4) array giving the values where:

Lutval (x,1) is the repetition count
Lutval (x,2) is the red value
Lutval (x,3) is the green value
Lutval (x,4) is the blue value

If the index is outside of the range set by PRASLR, the values are not changed at this location.

ENABLE/DISABLE RASTER VIDEO - RASTER ROUTINE

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASVI (OnOff, ErrHnd)
```

where:

OnOff is a LOGICAL
ErrHnd is the user-defined error-handler subroutine

DEFINITION

This subroutine is used to turn the Raster video on and off. (OnOff) is .TRUE. to turn the video on, and .FALSE. to turn the video off.

RASTER SUBROUTINES AND PARAMETERS

```
CALL PRASWP (Num, Pixval, ErrHnd)
```

where:

Num is an INTEGER*4
Pixval is an INTEGER*4 (Num,4) Array
ErrHnd is the user-defined error-handler subroutine

DEFINITION

This subroutine loads the current pixel location with the pixel values. (Num) specifies the number of entries in (Pixval). (Pixval) is an (Num,4) array where:

Pixval (x,1) is the repetition count
Pixval (x,2) is the red index
Pixval (x,3) is the green index
Pixval (x,4) is the blue index.

Name := RAWBLOCK

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRAWBL (Name, Size, Apply, ErrHnd)
```

where

Name is a CHARACTER STRING
Size is a INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine creates a structure consisting of a block of contiguous memory with a length of size bytes.

PS 300 COMMAND AND SYNTAX

Name := RAWBLOCK Size (APPLIED TO Apply);

Name := RATIONAL BSPLINE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRBspl (Name, Order, OpnCls, NonPer, Dimen,
&            NVert, Vertic, KntCnt, Knots,
&            Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING specifying the name to be given to the computed rational B-spline

Order is an INTEGER*4, specifying the order of the curve

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

For OpnCls .TRUE. is Open and .FALSE. is Closed

For NonPer .TRUE. is Non/periodic and .FALSE. is Periodic

NVert is an INTEGER*4 specifying the number of vertices

Vertic is defined: REAL*4 Vertic (4, NVert) specifying the vertices
where: Vertic(1,n) = x(n)
Vertic(2,n) = y(n)
Vertic(3,n) = z(n)
Vertic(4,n) = w(n)

Kntcnt is a INTEGER*4 specifying the number of knots

Knots is an array (KntCnt+1) of REAL*4 specifying the knot sequence

Chords is an INTEGER*4 specifying the number of vectors to be created

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine allows the parametric description of a rational B-spline curve form without having to specify or transfer the coordinates of each constituent vector.

(Continued on next page)

(continued)

PS 300 COMMAND AND SYNTAX

```
Name := RATIONAL BSPLINE
                ORDER = Order
                OPEN/CLOSED
                NONPERIODIC/PERIODIC
                N = NVert
                VERTICES  = X(1), Y(1), ( Z(1), ) W(1)
                            X(2), Y(2), ( Z(2), ) W(2)
                             :    :      :.
                            X(N), Y(N), ( Z(N), ) W(N)
                KNOTS = Knots (1), ... Knots (KntCnt)
                CHORDS = Chords;
```

NOTE

None of the parameters in the application subroutine PRBSPL are optional. The dimension must be specified in the PRBSPL application subroutine. In the PS 300 command, dimension is implied by syntax.

If KntCnt = 0, then the default knot sequence is generated and the knot array is ignored.

REMOVE NAME

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRem (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine removes (Name) from the display list.

PS 300 COMMAND AND SYNTAX

REMove Name;

REMOVE FOLLOWER of name

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRemFo (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine removes a previously placed 'follower' of (Name).

PS 300 COMMAND AND SYNTAX

REMove FOLLOWER of name;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRemFr (Name1, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING
Name2 is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine removes an instance of a named display data structure (Name1) from an instance node (Name2)

PS 300 COMMAND AND SYNTAX

REMove Name1 FROM Name2;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRemPr (Name, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine removes a previously placed prefix.

PS 300 COMMAND AND SYNTAX

REMove PREfix of name;

Name := ROTATE in X

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRotX  (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a 3x3 rotation matrix that rotates an object (Apply) around the x axis relative to world space origin. It has the following parametric definition:

- Angle is the x rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in X Angle (APPLied to Apply);

Name := ROTATE in Y

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRotY  (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a 3x3 rotation matrix that rotates an object (Apply) around the y axis relative to world space origin. It has the following parametric definition:

- Angle is the y rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in Y Angle (APPLied to Apply);

Name := ROTATE in Z

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRotZ  (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a 3x3 rotation matrix that rotates an object (Apply) around the z axis relative to world space origin. It has the following parametric definition:

- Angle is the z rotation angle in degrees

PS 300 COMMAND AND SYNTAX

Name := ROTate in Z Angle (APPLied to Apply);

Name := RATIONAL POLYNOMIAL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRPoly (Name, Order, Dimen, Coeffs, Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING

Order is an INTEGER*4

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

Coeffs is defined: REAL*4 Coeffs (4, Order+1)

```
                   where:  Coeffs (1,i) = x(order -i+1)
                           Coeffs (2,i) = y(order -i+1)
                           Coeffs (3,i) = z(order -i+1)
                           Coeffs (4,i) = w(order -i+1)
      :
      :
    etc.
```

To further clarify the description:

Coeffs(1,1) = the coefficient that will be applied to the t^{order} term

Coeffs(1,2) = the coefficient that will be applied to the $t^{order-1}$ term in the resultant x(t) function computed by this command.

Chords is an INTEGER*4

ErrHnd is the user-defined error-handler subroutine

(Continued on next page)

Name := RATIONAL POLYNOMIAL

(continued)

DESCRIPTION

This subroutine allows the parametric description of many curve forms without having to specify or transfer the coordinates of each constituent vector. It includes the following parametric definitions:

- Order is the order of the polynomial
- Coefficients represent the x,y,z components of the curve
- Chords is the number of vectors to be drawn

PS 300 COMMAND AND SYNTAX

```
Name := RATIONAL POLYNOMIAL
                ORDER = Order
                COEFFICIENTS  = X(i),   Y(i),   Z(i),   W(i)
                                X(i-1), Y(i-1), Z(i-1), W(i-1)
                                  :       :       :       :
                                X(0),   Y(0),   Z(0),   W(0)
                CHORDS = Chords;
```

RESERVE_WORKING_STORAGE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PRsvSt (Bytes, ErrHnd)
```

where:

Bytes is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine is used to reserve working storage space for rendering solids and surfaces. Working storage space must be reserved explicitly using this subroutine. The parameter (Bytes) represents the number of bytes to be reserved for working storage.

PS 300 COMMAND AND SYNTAX

Reserve_Working_Storage Bytes;

Name := SCALE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PScale (Name, V, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
V is defined: REAL*4 V(3)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine applies a scale transformation (V) to a specified vector list and/or characters (Apply). It contains the following parametric definition:

- V is a vector containing the x,y,z scale components
  V(1) = x scale factor
  V(2) = y scale factor
  V(3) = z scale factor

PS 300 COMMAND AND SYNTAX

Name := SCALE by V (APPLied to Apply);

NOTE

All three components must be specified in V.

Name := SET conditional_BIT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeBit (Name, BitNum, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
BitNum is an INTEGER*4
OnOff is .TRUE. for ON, .FALSE. for OFF
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine alters one of the 15 global conditional bits during the traversal of the data structure. These conditional bits are initially set to OFF. When the traversal is finished, the bits are restored to their previous values. It contains the following parametric definitions:

- BitNum is an integer from 0 to 14 corresponding to the conditional bit to be set to ON or OFF

PS 300 COMMAND AND SYNTAX

Name := SET conditional_BIT BitNum OnOff (APPLied to Apply);

Name := SET CHARACTERS SCREEN_oriented/FIXED

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeChF (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine sets the type of screen orientation for displayed character strings. When PSeChF is used, characters are not affected by rotation or scaling transformations and they are displayed with full size and intensity.

PS 300 COMMAND AND SYNTAX

Name := SET CHARacters SCREEN_oriented/FIXED (APPLied to Apply);

Name := SET CHARACTERS SCREEN_oriented

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeChS (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine sets the type of screen orientation for displayed character strings. When PSeChS is used, characters are not affected by rotation or scaling transformations, but intensity and size will still vary with depth (Z-position).

PS 300 COMMAND AND SYNTAX

Name := SET CHARacters SCREEN_oriented (APPLied to Apply);

Name: SET CHARACTERS WORLD_ORIENTED

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeChW (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine sets the type of screen orientation for displayed character strings. When PSeChW is used, characters are transformed along with any part of the object containing them.

PS 300 COMMAND AND SYNTAX

Name := SET CHARacters WORLD_oriented (APPLied to Apply);

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSECNS (Bool, Inp, Name, ErrHnd)
```

where

Bool is a LOGICAL
Inp is an INTEGER*4
Name is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine is used to define a particular function instance input to be a constant or trigger input.

PS 300 COMMAND AND SYNTAX

SETUP CNESS TRUE ‹Inp› Name;
SETUP CNESS FALSE ‹Inp› Name;

Name := SET COLOR

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeCol (Name, Hue, Sat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Hue is a REAL*4
Sat is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine specifies the color of an object (Apply). It contains the following parametric definition:

- Hue is greater than or equal to 0.0 and less than 360.0 with:
  0.0 = pure blue
  20.0 = pure red
  240.0 = pure green

- Sat is from 0.0 to 1.0 with:
  0.0 = no saturation (white)
  1.0 = full saturation

PS 300 COMMAND AND SYNTAX

Name := SET COLOR Hue,Sat (APPLied to Apply);

Name := SET CONTRAST

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeCon (Name, Contrast, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Contrast is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine changes the contrast of the data structure (Apply). It contains the following parametric definition:

- Contrast is from 0.0 to 1.0 with:
  0.0 = lowest contrast
  1.0 = highest contrast

PS 300 COMMAND AND SYNTAX

Name := SET CONTrast to Contrast (APPLied to Apply);

Name := SECTIONING_PLANE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSecPl (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING*(*)
Apply is a CHARACTER STRING*(*)
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine creates a sectioning-plane node designating a descendant polygon as a sectioning-plane. The parameter (Name) supplies the name to be given to the sectioning-plane operate node. (Apply) supplies the name of the entity that this node will be applied to.

PS 300 COMMAND AND SYNTAX

Name := SECTIONING_PLANE (Applied to Apply);

Name := SET CSM

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeCSM (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine allows the CSM to be set to ON or OFF; ON provides extra brightness and precision, OFF is the default setting and allows for the maximum number of vectors to be displayed.

PS 300 COMMAND AND SYNTAX

Name := SET CSM OnOff (APPLied to Apply);

Name := SET DISPLAYS ALL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeDAl (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sets all display(s) to ON or OFF.

PS 300 COMMAND AND SYNTAX

Name := SET DISPlays ALL OnOff (APPLied to Apply);

Name := SET DEPTH_CLIPPING

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeDCL (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine enables/disables depth clipping. With depth clipping Off, data between the front clipping plane and the eye will appear at full intensity and data behind the eye will be clipped.

PS 300 COMMAND AND SYNTAX

Name := SET DEPTH_CLipping OnOff (APPLied to Apply);

Name := SET DISPLAY

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeDOF (Name, OnOff, N, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
N is an INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies the display to be set to On or Off and has the following parametric definitions:

- N is the number of the display to be set to On or Off

PS 300 COMMAND AND SYNTAX

Name := SET DISPlay N OnOff (APPLied to Apply);

Name := SET INTENSITY

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeInt (Name, OnOff, IMin, IMax, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
IMin is a REAL*4
IMax is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies the intensity variation for depth cueing and has the following parametric definition:

- IMin is a real number from 0.0 to 1.0 that represents the dimmest intensity setting
- IMax is a real number from 0.0 to 1.0 that represents the brightest intensity setting.

PS 300 COMMAND AND SYNTAX

Name := SET INTENSITY OnOff IMin:IMax (APPLied to Apply);

Name := SET LEVEL_OF_DETAIL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeLOD (Name, Level, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Level is an INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine alters a global level of detail value temporarily. These temporary settings allow for conditional referencing to other data structures. When the traversal of data is finished, the level of detail is restored to its original level. It contains the following parametric definition:

- Level is an integer from 0 to 32767 that indicates the level of detail value

PS 300 COMMAND AND SYNTAX

Name := SET LEVel_of_detail TO Level (APPLied to Apply);

Name := SET PICKING IDENTIFIER

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSePID (Name, PickId, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
PickId is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies textual information that will be reported back if a pick occurs on the specified data structure (Apply). It contains the following parametric definition:

- PickId is the text that will be reported if a pick occurs anywhere within the structure (Apply)

PS 300 COMMAND AND SYNTAX

Name := SET PICKing IDentifier = PickId (APPLied to Apply);

Name := SET PICKING LOCATION

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSePLo (Name, XCentr, YCentr, Xsize, Ysize, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
XCentr, YCentr are REAL*4
Xsize, Ysize are REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies a rectangular picking area at (x,y) within the current viewport. It contains the following parametric definitions:

- XCentr, YCentr signify the center of the pick location
- Xsize, Ysize specify the boundaries of the pick rectangle

PS 300 COMMAND AND SYNTAX

Name := SET PICKING LOCation = XCentr, YCentr, Xsize, Ysize (APPLied to Apply);

Name := SET PLOTTER

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSEPLT (Name, Onoff, Apply, ErrHnd)
```

where

Name is a CHARACTER STRING
Onoff is a LOGICAL
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine enables or disables the plotting of subsequent nodes in the data structure.

PS 300 COMMAND AND SYNTAX

Name := SET PLOTTER Onoff (APPLIED TO Apply);

Name := SET PICKING OFF

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSePOF (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine enables/disables picking for a specified data structure (Apply).

PS 300 COMMAND AND SYNTAX

Name := SET PICKing OnOff (APPLied to Apply);

Name := SET RATE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeR (Name, PhaOn, PhaOff, IniOnF, Delay, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
PhaOn is an INTEGER*4
PhaOff is an INTEGER*4
IniOnF is a LOGICAL*1 defined: .TRUE. for On and .FALSE for Off
Delay is an INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sets two global duration values (PhaseOn and PhaseOff) during the traversal of a specified data structure (Apply). The default phase is off and never changes unless a SET RATE node is encountered. The subroutine has the following parametric definitions:

- PhaOn designates the duration of the ON phase
- PhaOff designates the duration of the OFF phase
- Delay is the number of refresh frames in the initial state

PS 300 COMMAND AND SYNTAX

Name := SET RATE PhaOn PhaOff IniOnF Delay (APPLied to Apply);

Name := SET RATE EXTERNAL

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSeREx (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine sets up a data structure that can be used to alter the Phase attribute using an external source, such as a function network or a message from the host computer.

PS 300 COMMAND AND SYNTAX

Name := SET RATE EXTernal (APPlied to Apply);

Name := SET COLOR BLENDING

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PsetCB (Name, Sat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Sat is REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine, when used with the ColorBlending parameter of the PVcLis subroutine, allows individual vector hue saturations to be set. It contains the following parametric definition:

- Sat is from 0.0 to 1.0 with:
  0.0 = no saturation (white)
  1.0 = full saturation

PS 300 COMMAND AND SYNTAX

Name := SET COLOR BLENDING Sat (Applied to Apply);

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnBoo (B, Inp, Dest, ErrHnd)
```

where:

B is .TRUE. or .FALSE., the logical value to be sent

*Inp is an INTEGER*4 corresponding to the input of the display data structure, function instance, or variable: Dest

Dest is a CHARACTER STRING representing the destination

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a Boolean value to input (Inp) of a specified function instance, display data structure, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND TRUE TO ‹Inp› Dest;
SEND FALSE TO ‹Inp› Dest;

* This mnemonic 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 inputs to display structures and their INTEGER*4 value is given below.

Mnemonic	‹Input›	INTEGER*4 Value
PILAST	‹LAST›	-5

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnFix (i, Inp, Dest, ErrHnd)
```

where:

i is an INTEGER*4, the integer to be sent

*Inp is an INTEGER*4 corresponding to the input of a display data structure, function instance, or variable

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends the value of (i) to the specified input (Inp) of the display data structure or function instance (Dest).

PS 300 COMMAND AND SYNTAX

SEND FIX (i) TO <Inp> Dest;

* 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 inputs to display structures and their INTEGER*4 value is given below.

Mnemonic	<Input>	INTEGER*4 Value
PIDEL	<DELETE>	-1
PICLR	<CLEAR>	-2

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnM2d (Mat, Inp, Dest, ErrHnd)
```

where:

Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)

Inp is an INTEGER*4 corresponding to the input of a variable, function instance or display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a 2x2 matrix to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND M2D (Mat) TO ‹Inp› Dest;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnM3d (Mat, Inp, Dest, ErrHnd)
```

where:

Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)

Inp is an INTEGER*4 corresponding to the input of a variable, function instance or display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a 3x3 matrix to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND M3D (Mat) TO ‹Inp› Dest;

SEND 4D MATRIX TO

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnM4d (Mat, Inp, Dest, ErrHnd)
```

where:

Mat is the matrix to be sent and is defined: REAL*4 Mat (4,4)

Inp is an INTEGER*4 corresponding to the input of a variable, function instance or display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a 4x4 matrix to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND M4D (Mat) TO <Inp> Dest;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnPL  (Count, DrawMv, Inp, Dest, ErrHnd)
```

where:

Count is an INTEGER*4

DrawMv is LOGICAL*1 and is defined: .TRUE. is Draw and .FALSE. is Move

Inp is an INTEGER*4

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine assigns Draw/Move specifications to consecutive vectors beginning at vector (Inp) of the vector list (Dest). It contains the following parametric definitions:

- Count is the number of Draws/Moves
- DrawMv is TRUE for Draw and FALSE for Move
- Inp corresponds to the index of the first vector to receive the Draw/Move specifications in the vector list (Dest)

PS 300 COMMAND AND SYNTAX

SEND Count*DrawMv TO ‹Inp› Dest;

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnRea (R, Inp, Dest, ErrHnd)
```

where:

R is the REAL*4 to be sent
Inp in an INTEGER*4
Dest is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a real number (R) to a specified input (Inp) of a display data structure or function instance (Dest).

PS 300 COMMAND AND SYNTAX

SEND Real-number TO ‹Inp› Dest;

SEND (RAW) 'Str' TO

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnRSt (Str, Inp, Dest, ErrHnd)
```

where:

Str is a CHARACTER STRING
*Inp is an INTEGER*4
Destination is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine does NOT translate the character string: Str. If the character string: Str = CHR(0) // CHR(1), then CHR(0) // CHR(1) is sent as the string. This subroutine is similar to the PSNST call, but no translation from EBCDIC to ASCII is performed on the string. This subroutine should be used when a character string of some length containing arbitrary characters is to be sent to a function network without translation.

An example of where PSNRST must be used is as follows.

Where the PS 300 command to send a string would be

```
SEND CHAR (1) to <2> CONSTANT1;
```

the equivalent Graphics Support Routine call would be

```
Str = Char (1)
CALL PSNRST (Str, 2, 'CONSTANT1', ErrHnd)
```

where Str is declared CHARACTER STRING*1

* These mnemonics may be referenced directly by the user if PROCONSF is INCLUDED in the subroutine. See the section on Programming Suggestions for a description of PROCONSF. A description of inputs to display structures and their INTEGER*4 value is given below.

Mnemonic	<Input>	INTEGER*4 Value
PILAST	<LAST>	-5
PISUBS	<SUBSTITUTE>	-6

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnSt  (Str, Inp, Dest, ErrHnd)
```

where:

Str is a CHARACTER STRING to be sent
*Inp is an INTEGER*4
Dest is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends the character string (Str) to the input (Inp) of a display data structure (Dest). The string (Str) is NOT translated from EBCDIC to ASCII.

PS 300 COMMAND AND SYNTAX

SEND 'Str' TO ‹Inp› Dest;

* 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 inputs to display data structures and their INTEGER*4 value is given below.

Mnemonic	‹Input›	INTEGER*4 Value
PILAST	‹LAST›	-5
PISUBS	‹SUBSTITUTE›	-6

SEND 2D VECTOR TO

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnV2d (V, Inp, Dest, ErrHnd)
```

where:

V is the vector to be sent and is defined: REAL*4 V(2)

*Inp is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a 2D vector to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND V2D (V) TO ‹Inp› Dest;

* 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 inputs to display data structures and their INTEGER*4 value is given below.

Mnemonic	‹Input›	INTEGER*4 Value
PIAPP	‹APPEND›	0
PISTEP	‹STEP›	-3
PIPOS	‹POSITION›	-4
PILAST	‹LAST›	-5

SEND 3D VECTOR TO

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnV3d (V, Inp, Dest, ErrHnd)
```

where:

V is the vector to be sent and is defined: REAL*4 V(3)

*Inp is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a 3D vector to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND V3D (V) TO <Inp> Dest;

* 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 inputs to display data structures and their INTEGER*4 value is given below.

Mnemonic	<Input>	INTEGER*4 Value
PIAPP	<APPEND>	0
PISTEP	<STEP>	-3
PIPOS	<POSITION>	-4
PILAST	<LAST>	-5

SEND 4D VECTOR TO

APPLICATION SUBROUTINE AND PARAMETERS

CALL PSnV4d (V, Inp, Dest, ErrHnd)

where:

V is the vector to be sent and is defined: REAL*4 V(4)

*Inp is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends a 4D vector to the specified input (Inp) of a display data structure, function instance, or variable (Dest).

PS 300 COMMAND AND SYNTAX

SEND V4D (V) TO ‹Inp› Dest;

* 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 inputs to display data structures and their INTEGER*4 value is given below.

Mnemonic	‹Input›	INTEGER*4 Value
PIAPP	‹APPEND›	0
PISTEP	‹STEP›	-3
PIPOS	‹POSITION›	-4
PILAST	‹LAST›	-5

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnVal (VarNam, Inp, Dest, ErrHnd)
```

where:

VarNam is a CHARACTER STRING that is the name of the Variable

*Inp is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display data structure

Dest is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine sends the current value in the variable (VarNam) to a designated input (Inp) of a display data structure or function instance (Dest).

PS 300 COMMAND AND SYNTAX

SEND VALUE (VarNam) TO ‹Inp› Dest;

* 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 inputs to display data structures and their INTEGER*4 value is given below.

Mnemonic	‹Input›	INTEGER*4 Value
PIAPP	‹APPEND›	0
PIDEL	‹DELETE›	-1
PICLR	‹CLEAR›	-2
PISTEP	‹STEP›	-3
PIPOS	‹POSITION›	-4
PILAST	‹LAST›	-5
PISUBS	‹SUBSTITUTE›	-6

SEND VECTOR LIST

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSnVL  (Name1, Inp, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING containing the name of the Vector list to be sent

*Inp is an INTEGER*4 corresponding to the index of the first vector to be replaced in (Name2) with the vectors from (Name1)

Name2 is a CHARACTER STRING containing the name of the destination of the Vector list

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine replaces the vectors beginning at vector (Inp) of the vector list (Name2) with the vectors from vector list (Name1).

PS 300 COMMAND AND SYNTAX

SEND VL (Name1) TO ‹Inp› Name2;

* This mnemonic 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 inputs to display data structures and their INTEGER*4 value is given below.

Mnemonic	‹Input›	INTEGER*4 Value
PIAPP	‹APPEND›	0
PILAST	‹LAST›	-5

Name := SOLID_RENDERING

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSolRe (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING*(*)
Apply is a CHARACTER STRING*(*)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines a solid-rendering operate node, marking its descendent structure so that solid renderings can be performed on it. The parameter (Name) supplies the name to be given to the solid-rendering operate node. (Apply) supplies the name of the entity that this operate node will be applied to.

PS 300 COMMAND AND SYNTAX

Name := SOLID_RENDERING (Applied to Apply);

Name := STANDARD FONT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PStdFo (Name, Apply, ErrHnd)
```

where:

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

DESCRIPTION

This subroutine establishes the standard PS 300 character font as the working font.

PS 300 COMMAND AND SYNTAX

Name := STANdard FONT (APPLied to Apply);

Name := SURFACE_RENDERING

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSurRe (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING*(*)
Apply is a CHARACTER STRING*(*)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines a surface-rendering operate node, marking its descendent structure so that surface renderings can be performed on it. The parameter (Name) supplies the name to be given to the surface-rendering operate node. (Apply) supplies the name of the entity that this operate node will be applied to.

PS 300 COMMAND AND SYNTAX

Name := SURFACE_RENDERING (Applied to Apply);

Name := TRANSLATE

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PTrans (Name, V, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING

V is the vector containing the x,y,z translation values and is defined: REAL*4 V(3)

Apply is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine applies a translation vector (V) to the specified data structure (Apply).

V(1) = x translation
V(2) = y translation
V(3) = z translation

PS 300 COMMAND AND SYNTAX

Name := TRANslate by V (APPLied to Apply);

NOTE

All 3 components in V must be specified. Z is not optional.

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVar (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING containing the name of the variable to be created.

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine defines a PS 300 variable where (Name) contains the name of the variable to be created.

PS 300 COMMAND AND SYNTAX

VARiable Name;

Name := VECTOR_LIST (no corresponding command)

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) 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)

(continued)

* 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

Name := VECTOR_LIST (no corresponding command)

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVcEnd (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to end 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) N=n <vectors>;

Name := VECTOR_LIST (no corresponding command)

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
0 <= Vecs(4,n) <=1

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. For block-normalized vector lists, this subroutine can only be called once. Multiple calls to this subroutine are not permitted for the Block-normalized vector list case. 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)

Name := VECTOR_LIST (no corresponding command)

(continued)

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

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

NOTE

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, 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.

Name := VIEWPORT

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PViewP (Name, XMin, XMax, YMin, YMax, IMin, IMax, Apply,
             ErrHnd)
```

where:

Name is a CHARACTER STRING
XMin, Xmax (horizontal) are REAL*4
YMin, Ymax (vertical) are REAL*4
IMin, IMax are REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It contains the following parametric definitions:

- XMin, Xmax (horizontal) specify the horizontal boundaries of the new viewport
- YMin, Ymax (vertical) specify the vertical boundaries of the new viewport
- IMin, IMax specify the minimum and maximum intensities for the viewport

PS 300 COMMAND AND SYNTAX

```
Name := VIEWport  HORizontal  = Xmin:Xmax
                  VERTical    = Ymin:Ymax
                  INTENsity   = Imin:Imax
        (APPLied to Apply);
```

Name := WINDOW

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PWindo (Name, Xmin, Xmax, Ymin, Ymax, Front, Back, Apply,
             ErrHnd)
```

where:

Name is a CHARACTER STRING
XMin, Xmax (horizontal) are REAL*4
YMin, Ymax (vertical) are REAL*4
Front is a REAL*4
Back is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies a right rectangular prism enclosing a portion of the data space to be displayed in parallel projection. It contains the following parametric definitions:

- XMin, Xmax (horizontal) specify the window's boundaries along the x axis
- YMin, Ymax (vertical) specify the window's boundaries on the y axis
- Front specifies the front boundary
  Back specifies the back boundary

PS 300 COMMAND AND SYNTAX

```
Name := WINDOW X = Xmin:Xmax
              Y = Ymin:Ymax
              FRONT boundary = Front
              BACK  boundary = Back
              (APPLied to Apply);
```

Name := CANCEL XFORM

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PXFCAN (Name, Apply, ErrHnd)
```

where

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

DESCRIPTION

This subroutine stops transform data processing of subsequent nodes.

PS 300 COMMAND AND SYNTAX

Name := CANCEL XFORM (APPLIED TO Apply);

Name := XFORM MATRIX

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PXFMAT (Name, Apply, ErrHnd)
```

where

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

DESCRIPTION

This subroutine allows subsequent nodes to be processed to produce a transformation matrix.

PS 300 COMMAND AND SYNTAX

Name := XFORM MATRIX (APPLIED TO Apply);

Name := XFORM VECTOR_LIST

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PXFVEC (Name, Apply, ErrHnd)
```

where

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

DESCRIPTION

This subroutine allows subsequent nodes to be processed to produce a transformed vector_list.

PS 300 COMMAND AND SYNTAX

Name := XFORM VECTOR_LIST (APPLIED TO Apply);

PS 340 GSR FORTRAN ERROR CODE DEFINITIONS

The following tables listed in this section define the possible error codes used to identify warning or error conditions that may arise while using the Graphics Support Routines. The set of possible error codes is divided into several regions reserved for specific severity and machine dependency levels.

1...255	= Machine INDEPENDENT warning conditions.
256...511	= Machine DEPENDENT warning conditions.
512...767	= Machine INDEPENDENT error conditions.
768...1023	= Machine DEPENDENT error conditions.
1024...1279	= Machine INDEPENDENT fatal error conditions.
1280...1535	= Machine DEPENDENT fatal error conditions.

ERROR TABLE - 1

The following warning codes allow successful completion of the GSR subroutine, but indicate a probable user error.

Error Code	Mnemonic	Severity	Meaning
1	PSWBNC:	Warning:	Bad name character. Any invalid PS 300 name is translated to the underscore character.
2	PSWNTL:	Warning:	Name too long. Name truncated to 256 characters.
3	PSWSTL:	Warning:	String too long. String truncated to 240 characters.
30	PSWPCG:	Warning:	The Pixel Count is greater than the screen size in call to PRASWP. (Reserved for P6.V01 Raster subroutines.)
31	PSWPCL:	Warning:	The Pixel Count is less than 1 in call to PRASWP. (Reserved for P6.V01 Raster subroutines.)
32	PSWRCG:	Warning:	Repetition count greater than 255 in call to PRASLU. (Reserved for P6.V01 Raster subroutines.)
33	PSWRCL:	Warning:	Repetition count less than 1 in call to PRASLU. (Reserved for P6.V01 Raster subroutines.)

Error Code	Mnemonic	Severity	Meaning
256	PSWAAD:	Warning:	Attach already done.
257	PSWAKS:	Warning:	Attention key seen. This tells the error-handling routine that the user hit the Attention key (IBM version only).
258	PSWBGC:	Warning:	The string specified to be sent to the "generic" output channel of CIROUTE via the PPutGX subroutine contained an invalid character that has been translated to a blank space character. This error code CANNOT be caused by invoking the subroutine: PPutG which does NOT perform any translation on the specified string (IBM version only).
259	PSWBSC:	Warning:	Bad string character. Any invalid string character is converted to a blank space character.
260	PSWBPC:	Warning:	The string specified to be sent to the PS 300 Parser via the PPutP subroutine contained an invalid character that has been translated to a blank space character.

ERROR TABLE - 2

For the following errors, the GSRs abort the current command sequence (if there is one) and ignore the out-of-sequence command that (probably) caused this error.

Error Code	Mnemonic	Severity	Meaning
515	PSEPOE:	Error:	Prefix operate node call expected.
516	PSEFOE:	Error:	Follow operate node call expected.
517	PSELBE:	Error:	Label block call expected.
518	PSEVLE:	Error:	Vector List call expected.
519	PSEAMV:	Error:	Attempted multiple PVcLis call sequence for block normalized vectors prohibited.

Error Code	Mnemonic	Severity	Meaning
520	PSEMLB:	Error:	Missing label block begin call.
521	PSEMVB:	Error:	Missing vector list begin call.
529	PSEMPB:	Error:	The Begin polygon call is missing. PPlygA, PPlygL, or PPlygE was called without the prerequisite call to PPlygB.
530	PSEALE:	Error:	A call to PPlygA, PPlygL, or PPlygE was expected.
531	PSELEX:	Error:	A call to PPlygL or PPlygE was expected.
532	PSEALX:	Error:	A call to PPlygA or PPlygL was expected.
533	PSELX:	Error:	A call to PPlygL was expected.

ERROR TABLE - 3

The following errors are user errors and are generated by invalid parameters or by an unsuccessful attempt to attach.

Error Code	Mnemonic	Severity	Meaning
512	PSEIMC:	Error:	Invalid multiplexing channel argument specified in a call to PMuxP, PMuxCI, or PMuxG. The multiplexing channel assigned to the Parser, CI, or Generic channel is not changed.
513	PSEIVC:	Error:	Invalid vector list class specified in call to PVcBeg. Command is ignored.
514	PSEIVD:	Error:	Invalid vector list dimension specified in call to PVcBeg. Command is ignored.
522	PSENUN:	Error:	A null name is not permitted in this call context. The command is ignored.
523	PSEBCT:	Error:	Bad Comparison type operator specified. If Level = command ignored.

Error Code	Mnemonic	Severity	Meaning
524	PSEIFN:	Error:	Attempted PS 300 function instance call failed because the named function cannot possibly exist. The function name identifying the function type to instance was longer than 256 characters.
525	PSENNR:	Error:	Null name was required for parameter in operate node call following a PPref or PFoll subroutine.
526	PSETME:	Error:	Too many PEnds calls for the number of preceding PBegs calls. Command ignored.
527	PSENOA:	Error:	The PS 300 communications link has not been established. The user failed to call PAttch or an error occurred in the attach procedure preventing the communications link from being created.
528	PSEODR:	Error:	An overrun occurred during a read operation. The user-supplied input buffer was too small and truncation has occurred.
534	PSEMPX:	Error:	The polygon specified by the call to PPlygL contains more than 250 vertices. The polygon is ignored.
535	PSELMP:	Error:	The polygon specified by the cll to PPlygL contains fewer than 3 vertices. It is therefore a degenerate polygon and is ignored.
536	PSEIPA:	Error:	Illegal polygon attribute(s) specified in the call to PPlygA. The attribute(s) are ignored.
550	PSEICP:	Error:	Illegal Current Pixel specification in call to PRASCP. (Reserved for P6.V01 Raster subroutines.)
552	PSEIOR:	Error:	Index out of range: 0...255 in call to PRASLU. (Reserved for P6.V01 Raster subroutines.)
553	PSELDC:	Error:	Illegal LDC specification in call to PRASLD. (Reserved for P6.V01 Raster subroutines.)

Error Code	Mnemonic	Severity	Meaning
554	PSELNL:	Error:	NUM parameter less than 1 in call to PRASLD. (Reserved for P6.V01 Raster subroutines.)
555	PSEMGM:	Error:	Minimum > Maximum in call to PRASLR. (Reserved for P6.V01 Raster subroutines.)
556	PSEMNO:	Error:	Minimum out of range 0...255 in call to PRASLR. (Reserved for P6.V01 Raster subroutines.)
557	PSEMXO:	Error:	Maximum out of range 0...255 in call to PRASLR. (Reserved for P6.V01 Raster subroutines.)
558	PSEPNL:	Error:	NUM parameter less than 1 in call to PRASWP. (Reserved for P6.V01 Raster subroutines.)

At the present time, the following three error messages (780, 781, 782) are only meaningful for Digital Equipment Corporation (DEC) VAX/VMS*. All three errors indicate that the parameter passed as a string in PAttch was not successfully parsed and that the Attach call failed.

Error Code	Mnemonic	Severity	Meaning
780	PSEPDT:	Error:	This error indicates that a missing or invalid Physical Device Type was specified in a call to PAttch.
781	PSELDN:	Error:	This error indicates that a missing or invalid Logical Device Name was specified in a call to PAttch.
782	PSEADE:	Error:	This error indicates that an Attach delimiter was expected in a call to PAttch.

*Trademark of the Digital Equipment Corporation, Maynard, Massachusetts

FATAL ERRORS

The errors listed below indicate a very serious error condition. If the user's error handler is invoked with any of the error codes listed below, the program execution should be aborted.

ERROR TABLE - 4

Error Code	Mnemonic	Severity	Meaning
1024	PSFIFC:	Fatal Error:	Illegal frame command specified in call to PSUTIL_RasMode. This error code indicates an internal validity check error. E&S Software Support should be contacted.
1280	PSFPAF:	Fatal Error:	Physical Attach operation failed.
1281	PSFPDF:	Fatal Error:	Physical Detach operation failed.
1282	PSFPGF:	Fatal Error:	Physical Get operation failed.
1283	PSFPPF:	Fatal Error:	Physical Put operation failed.

The following three errors are only applicable to the DEC VAX/VMS version of the Graphics Support Routines. All three error codes indicate an internal Graphics Support Routines validity error. E&S Software Support should be contacted if these errors are detected.

Error Code	Mnemonic	Severity	Meaning
1290	PSFBTL:	Fatal Error:	Buffer too large in a call to PSPUT. Internal validity check error.
1291	PSFWNA:	Fatal Error:	Wrong number of arguments to low-levelI/O subroutine in PROIOLIB.MAR. Validity error.
1292	PSFPTL:	Fatal Error:	Prompt too large in call to PSFRCV. Internal validity check error.

APPENDIX A. SAMPLE PROGRAMS

This appendix contains sample FORTRAN programs that illustrate the use of the PS 300 DEC/VAX FORTRAN-77 Graphics Support Routines. The programs each contain examples of an error-handler subroutine.

This is a FORTRAN vector list example program

```
      PROGRAM CircleTest

      INCLUDE 'PROCONST.FOR/NOLIST'

      REAL*4    Circle_List (4, 101)
      INTEGER*4 Dimensionality, Class
      CHARACTER ClassType*1, Mode*1, DeviceSpec*1, DeviceName*5,
     &          AttachParameter*80
      LOGICAL   BlockNormalized
      LOGICAL*1 MoveDraw (101)

      CHARACTER Uppercase*1
      EXTERNAL  ERR, Uppercase

      DeviceSpec = ' '
      DO WHILE ((DeviceSpec .NE. 'A') .AND.
     &          (DeviceSpec .NE. 'D') .AND.
     &          (DeviceSpec .NE. 'P'))
        WRITE (6, 1) 'Device Interface type = '
     &             // '(Asynchronous, DMR-11, '
     &             // 'Parallel): _'
```

```
      READ (5, 2) DeviceSpec
      DeviceSpec = Uppercase (DeviceSpec)
      IF ((DeviceSpec .NE. 'A') .AND.
     &    (DeviceSpec .NE. 'D') .AND.
     &    (DeviceSpec .NE. 'P')) THEN
        WRITE (6, *) 'Invalid device type specified.'
      END IF
     END DO
     DeviceName = ' '
     DO WHILE (DeviceName .EQ. ' ')
      WRITE (6, 1) 'Physical Device name (i.e. '
     &         // 'TT, TTA6, XMD0, PIA0): _'
      READ (5, 3) DeviceName
     END DO
     IF ((DeviceName (2:2)) .EQ. ' ') THEN
      DeviceName (2:) = ':'
     ELSE
      IF ((DeviceName (3:3)) .EQ. ' ') THEN
       DeviceName (3:) = ':'
      ELSE
       IF ((DeviceName (4:4)) .EQ. ' ') THEN
        DeviceName (4:) = ':'
       ELSE
        DeviceName (5:) = ':'
       END IF
      END IF
     END IF
     WRITE (6, 1) 'Vector mode = (Block, Vector): _'
     READ (5, 2) Mode
     IF ((Uppercase (Mode)) .EQ. 'B') THEN
      BlockNormalized = .TRUE.
     ELSE
      BlockNormalized = .FALSE.
     END IF
     WRITE (6, 1) 'Dimensionality = (2, 3): _'
     READ (5, *) Dimensionality
     WRITE (6, 1) 'Class = (Connected, Dots, '
    &         // 'Itemized, Separate): _'
    1 FORMAT(' ', A, $)
     READ (5, 2) ClassType
    2 FORMAT(1A)
    3 FORMAT(5A)
     IF ((Uppercase (ClassType)) .EQ. 'C') THEN
      Class = PVConn
     ELSE IF ((Uppercase (ClassType)) .EQ. 'D') THEN
      Class = PVDots
     ELSE IF ((Uppercase (ClassType)) .EQ. 'I') THEN
      Class = PVItem
     ELSE IF ((Uppercase (ClassType)) .EQ. 'S') THEN
      Class = PVSepa
```

```
 ELSE
   Class = PVConn
 END IF
 IF ((Uppercase (DeviceSpec)) .EQ. 'D') THEN
   AttachParameter = 'Logdevnam=' // DeviceName
&              // '/Phydevtyp=DMR-11'
 ELSE IF ((Uppercase (DeviceSpec)) .EQ. 'P') THEN
   AttachParameter = 'Logdevnam=' // DeviceName
&              // '/Phydevtyp=Parallel'
 ELSE
   AttachParameter = 'Logdevnam=' // DeviceName
&              // '/Phydevtyp=Async'
 END IF
 CALL Pattch (AttachParameter, ERR)
 CALL PInit  ( Err )
 CALL Circle (Circle_List, MoveDraw)
 CALL Pvcbeg ('A', 101, BlockNormalized, .FALSE.,
&           Dimensionality, Class, Err)
 CALL Pvclis ( 101, circle_list, MoveDraw, Err )
 CALL Pvcend ( err )
 CALL Pdisp ('A', err )
 CALL Pdtach ( err )
 END
```

SUBROUTINE Circle (Circle_List, MoveDraw)

```
REAL*4    Circle_List (4, *)
LOGICAL*1  MoveDraw (*)

REAL*4    Deg_rad
PARAMETER  (Deg_rad    = 0.017453292)

REAL*4    Theta, DTheta
INTEGER*4  i
LOGICAL    Draw

Draw = .FALSE.
DTheta = 3.6 * Deg_Rad
Theta = 0
DO i = 1, 101
  circle_list (1, i) = 0.8 * cos ( theta )
  circle_list (2, i) = 0.8 * sin ( theta )
  circle_list (3, i) = 0
  circle_list (4, i) = 1
  MoveDraw (i) = Draw
  Theta = Theta + DTheta
  Draw = .NOT. Draw
END DO
RETURN
END
```

```
      CHARACTER*1 FUNCTION Uppercase (Chara)
      CHARACTER   Chara*(*)
      IF (((Chara (1:1)) .GE. 'a') .AND.
     &    ((Chara (1:1)) .LE. 'z')) THEN
        Uppercase = CHAR (ICHAR (Chara (1:1)) - 32)
      ELSE
        Uppercase = Chara
      END IF
      RETURN
      END



C
C
C
C  The following Error Handler demonstrates the general
C  overall recommended form that the user's own error
C  handler should follow.
C
C  This error handler upon being invoked writes ALL
C  messages to the data file: 'PROERROR.LOG'. Error
C  and warning explanation messages are written to
C  a data file for 2 reasons:
C
C
C     1. The error handler should NOT immediately
C        write information out on the PS 300 screen
C        since the explanatory text defining the error
C        or warning condition may be taken as data by
C        the PS 300 and therefore wind up not being
C        displayed on the PS 300 screen (as in the
C        case of a catastrophic data transmission
C        error).
C
C     2. The logging of errors and warnings to a
C        logfile allows any errors and/or warnings
C        to be reviewed at a later time.
C
C
C
```

```
      SUBROUTINE ERR ( ERRCOD )

C
C     Procedural Interface (GSR) error handler:
C

      INCLUDE  'PROCONST.FOR/NOLIST'
      INTEGER*4  ERRCOD
      INTEGER*4  PsVMSerr
      LOGICAL   FILOPN
      DATA      FILOPN /.FALSE./
      EXTERNAL   PsVMSerr, DETERH, PIDCOD

      IF (FILOPN) GOTO 1
C
C     Open error file for logging of errors:
C
        OPEN (UNIT=10, FILE='PROERROR.LOG', STATUS='NEW',
     &      DISP='KEEP', ORGANIZATION='SEQUENTIAL',
     &      ACCESS='SEQUENTIAL', CARRIAGECONTROL='LIST')
        FILOPN = .TRUE.
C     END IF
    1 CALL PIDCOD (ERRCOD)
      IF (ERRCOD .LT. 512) GOTO 3
        WRITE (10, *) 'PS-I-ATDCOMLNK:  Attempting to '
     &          // 'detach PS 300/Host communications '
     &          // 'link.'
C
C     When we attempt to perform the Detach, use a
C     different error handler so as not to get caught
C     in a recursive loop if we consistently get an
C     error when attempting to detach.
C
        CALL PDTACH (DETERH)
        CLOSE (UNIT=10)
        IF ((ERRCOD .LT. PSFPAF) .OR.
     &     (ERRCOD .GT. PSFPPF)) GOTO 2
C
C       Identify VMS error if there was one
C
        CALL LIB$STOP (%VAL (PsVMSerr ()))
        GOTO 3
C     ELSE
    2   STOP
C     END IF
C     END IF
    3 RETURN
      END
```

```
      SUBROUTINE DETERH (ERRCOD)

C
C     Main Error handler Detach error handler:
C

      INTEGER*4  ERRCOD
      EXTERNAL   PIDCOD

      WRITE (10, *) 'PS-I-ERRWARDET:  Error/warning '
     &        // 'trying to Detach '
     &        // 'the communications'
      WRITE (10, *) 'link between the PS 300 and the host.'
      CALL PIDCOD (ERRCOD)
      RETURN
      END


      SUBROUTINE PIDCOD (ERRCOD)

C
C     PIDCOD: Identify Procedural Interface (GSR) Completion
C          code.
C

      INCLUDE  'PROCONST.FOR/NOLIST'
      INTEGER*4  ERRCOD
      CHARACTER  VMSDEF*133, PIDEF*133
      INTEGER*4  PsVMSerr
      CHARACTER  MSSG1*55, MSSG2*67
      PARAMETER  (MSSG1 = 'PS-W-UNRCOMCOD:  Procedural '
     &            // 'Interface '
     &            // '(GSR) completion ')
      EXTERNAL   PsVMSerr

      WRITE (10, *) 'PS-I-PROERRWAR:  Procedural '
     &        // 'Interface warning/'
     &        // 'error completion code was '
      WRITE (10, *) 'received.'
      IF (ERRCOD .NE. PSWBNC) GOTO 1
        WRITE (10, *) 'PS-W-BADNAMCHR:  Bad character '
     &          // 'in name was '
     &          // 'translated to:  "_".'
        GOTO 1000
C     ELSE
    1 IF (ERRCOD .NE. PSWNTL) GOTO 2
```

```
      WRITE (10, *) 'PS-W-NAMTOOLON: Name too '
     &          // 'long. Name was '
     &          // 'truncated to '
      WRITE (10, *) '256 characters.'
      GOTO 1000
C     ELSE
    2 IF (ERRCOD .NE. PSWSTL) GOTO 7
      WRITE (10, *) 'PS-W-STRTOOLON: String too '
     &          // 'long. String '
     &          // 'was truncated '
      WRITE (10, *) 'to 240 characters.'
      GOTO 1000
C     ELSE
    7 IF (ERRCOD .NE. PSWAAD) GOTO 8
      WRITE (10, *) 'PS-W-ATTALRDON: Attach '
     &          // 'already done. '
     &          // 'Multiple call to PAttch without'
      WRITE (10, *) 'intervening PDtach call ignored.'
      GOTO 1000
C     ELSE
    8 IF (ERRCOD .NE. PSWAKS) GOTO 9
      WRITE (10, *) 'PS-W-ATNKEYSEE: Attention key '
     &          // 'seen (depressed).'
      CALL PIBMSP
      GOTO 1000
C     ELSE
    9 IF (ERRCOD .NE. PSWBGC) GOTO 10
      WRITE (10, *) 'PS-W-BADGENCHR: Bad generic '
     &          // 'channel character. Bad '
      WRITE (10, *) 'character in string sent via: '
     &          // 'PPutGX  was translated to '
      WRITE (10, *) 'a blank.'
      CALL PIBMSP
      GOTO 1000
C     ELSE
   10 IF (ERRCOD .NE. PSWBSC) GOTO 11
      WRITE (10, *) 'PS-W-BADSTRCHR: Bad '
     &          // 'character in string was '
     &          // 'translated to a blank.'
      CALL PIBMSP
      GOTO 1000
C     ELSE
   11 IF (ERRCOD .NE. PSWBPC) GOTO 12
      WRITE (10, *) 'PS-W-BADPARCHR: Bad parser '
     &          // 'channel character. Bad '
     &          // 'character in string sent to'
      WRITE (10, *) 'PS 300 parser via: PPutP  '
     &          // 'was translated to a blank.'
      CALL PIBMSP
      GOTO 1000
```

```
C     ELSE
   12 IF (ERRCOD .NE. PSEIMC) GOTO 13
         WRITE (10, *) 'PS-E-INVMUXCHA:  Invalid '
     &           // 'multiplexing channel '
     &           // 'specified in call to:'
         WRITE (10, *) 'PMuxCI, PMuxP, or PMuxG.'
         GOTO 1000
C     ELSE
   13 IF (ERRCOD .NE. PSEIVC) GOTO 14
         WRITE (10, *) 'PS-E-INVVECCLA:  Invalid '
     &           // 'vector list class '
     &           // 'specified'
         WRITE (10, *) 'in call to:  PVcBeg.'
         GOTO 1000
C     ELSE
   14 IF (ERRCOD .NE. PSEIVD) GOTO 15
         WRITE (10, *) 'PS-E-INVVECDIM:  Invalid '
     &           // 'vector list dimension '
     &           // 'specified in call to'
         WRITE (10, *) 'PVcBeg.'
         GOTO 1000
C     ELSE
   15 IF (ERRCOD .NE. PSEPOE) GOTO 16
         WRITE (10, *) 'PS-E-PREOPEEXP:  Prefix '
     &           // 'operator call was '
     &           // 'expected.'
         GOTO 1000
C     ELSE
   16 IF (ERRCOD .NE. PSEFOE) GOTO 17
         WRITE (10, *) 'PS-E-FOLOPEEXP:  Follow '
     &           // 'operator call was '
     &           // 'expected.'
         GOTO 1000
C     ELSE
   17 IF (ERRCOD .NE. PSELBE) GOTO 18
         WRITE (10, *) 'PS-E-LABBLKEXP:  Call to '
     &           // 'PLaAdd or PLaEnd was '
     &           // 'expected.'
         GOTO 1000
C     ELSE
   18 IF (ERRCOD .NE. PSEVLE) GOTO 19
         WRITE (10, *) 'PS-E-VECLISEXP:  Call to '
     &           // 'PVcLis or PVcEnd '
     &           // 'was expected.'
         GOTO 1000
```

```
C     ELSE
   19 IF (ERRCOD .NE. PSEAMV) GOTO 20
         WRITE (10, *) 'PS-E-ATTMULVEC:  Attempted '
     &         // 'multiple call '
     &         // 'sequence to PVcLis is NOT'
         WRITE (10, *) 'permitted for BLOCK '
     &         // 'normalized vectors.'
         GOTO 1000
C     ELSE
   20 IF (ERRCOD .NE. PSEMLB) GOTO 21
         WRITE (10, *) 'PS-E-MISLABBEG:  Missing '
     &         // 'label block begin call.  '
     &         // 'Call to PLaAdd or PLaEnd'
         WRITE (10, *) 'without call to:  PLaBeg.'
         GOTO 1000
C     ELSE
   21 IF (ERRCOD .NE. PSEMVB) GOTO 22
         WRITE (10, *) 'PS-E-MISVECBEG:  Missing '
     &         // 'vector list begin '
     &         // 'call.  Call to PVcLis'
         WRITE (10, *) 'or PVcEnd without call '
     &         // 'to:  PVcBeg.'
         GOTO 1000
C     ELSE
   22 IF (ERRCOD .NE. PSENUN) GOTO 23
         WRITE (10, *) 'PS-E-NULNAM:  Null name '
     &         // 'parameter is not allowed.'
         GOTO 1000
C     ELSE
   23 IF (ERRCOD .NE. PSEBCT) GOTO 24
         WRITE (10, *) 'PS-E-BADCOMTYP:  Bad '
     &         // 'comparison type operator '
     &         // 'specified in '
         WRITE (10, *) 'call to:  PIfLev.'
         GOTO 1000
C     ELSE
   24 IF (ERRCOD .NE. PSEIFN) GOTO 25
         WRITE (10, *) 'PS-E-INVFUNNAM:  Invalid '
     &         // 'function name.  '
     &         // 'Attempted PS 300'
         WRITE (10, *) 'function instance failed '
     &         // 'because the named '
     &         // 'function cannot possibly'
         WRITE (10, *) 'exist.  The function name '
     &         // 'identifying the '
     &         // 'function type to instance'
         WRITE (10, *) 'was longer than 256 characters.'
         GOTO 1000
```

```
C     ELSE
   25 IF (ERRCOD .NE. PSENNR) GOTO 26
         WRITE (10, *) 'PS-E-NULNAMREQ:  Null name '
     &           // 'parameter is '
     &           // 'required in operate node'
         WRITE (10, *) 'call following a PPref or '
     &           // 'PFoll procedure call.'
         GOTO 1000
C     ELSE
   26 IF (ERRCOD .NE. PSETME) GOTO 27
         WRITE (10, *) 'PS-E-TOOMANEND:  Too '
     &           // 'many END_STRUCTURE calls '
     &           // 'invoked.'
         GOTO 1000
C     ELSE
   27 IF (ERRCOD .NE. PSENOA) GOTO 28
         WRITE (10, *) 'PS-E-NOTATT:  The PS 300 '
     &           // 'communications link '
     &           // 'has not '
         WRITE (10, *) 'yet been established.  '
     &           // 'PAttch has not been '
     &           // 'called or failed.'
         GOTO 1000
C     ELSE
   28 IF (ERRCOD .NE. PSEODR) GOTO 29
         WRITE (10, *) 'PS-E-OVEDURREA:  An '
     &           // 'overrun occurred during '
     &           // 'a read operation.'
         WRITE (10, *) 'The specified input buffer '
     &           // 'in call to:  PGET  '
     &           // 'or:  PGETW'
         WRITE (10, *) 'was too small and '
     &           // 'truncation has occurred.'
         GOTO 1000
C     ELSE
   29 IF (ERRCOD .NE. PREICP) GOTO 38
   38 IF (ERRCOD .NE. PSEPDT) GOTO 39
         WRITE (10, *) 'PS-E-PHYDEVTYP:  Missing '
     &           // 'or invalid physical '
     &           // 'device type'
         WRITE (10, *) 'specifier in call to PAttch.'
         CALL PVAXSP
         GOTO 1000
C     ELSE
   39 IF (ERRCOD .NE. PSELDN) GOTO 40
         WRITE (10, *) 'PS-E-LOGDEVNAM:  Missing '
     &           // 'or invalid logical '
     &           // 'device name'
         WRITE (10, *) 'specifier in call to PAttch.'
         CALL PVAXSP
         GOTO 1000
```

```
C     ELSE
   40 IF (ERRCOD .NE. PSEADE) GOTO 41
         WRITE (10, *) 'PS-E-ATTDELEXP:  Attach '
     &           // 'parameter string '
     &           // 'delimiter'
         WRITE (10, *) '"/" was expected.'
         CALL PVAXSP
         GOTO 1000
C     ELSE
   41 IF (ERRCOD .NE. PSFPAF) GOTO 42
         WRITE (10, *) 'PS-F-PHYATTFAI:  '
     &           // 'Physical attach operation '
     &           // 'failed.'
         GOTO 1000
C     ELSE
   42 IF (ERRCOD .NE. PSFPDF) GOTO 43
         WRITE (10, *) 'PS-F-PHYDETFAI:  Physical '
     &           // 'detach operation '
     &           // 'failed.'
         GOTO 1000
C     ELSE
   43 IF (ERRCOD .NE. PSFPGF) GOTO 44
         WRITE (10, *) 'PS-F-PHYGETFAI:  Physical '
     &           // 'GET operation failed.'
         GOTO 1000
C     ELSE
   44 IF (ERRCOD .NE. PSFPPF) GOTO 45
         WRITE (10, *) 'PS-F-PHYPUTFAI:  Physical '
     &           // 'PUT operation failed.'
         GOTO 1000
C     ELSE
   45 IF (ERRCOD .NE. PSFBTL) GOTO 46
         WRITE (10, *) 'PS-F-BUFTOOLAR:  Buffer '
     &           // 'too large error in '
     &           // 'call to:  PSPUT.'
         WRITE (10, *) 'This error should NEVER '
     &           // 'occur and indicates a '
     &           // 'Procedural Interface (GSR)'
         WRITE (10, *) 'internal validity check.'
         CALL PVAXSP
         GOTO 1000
C     ELSE
   46 IF (ERRCOD .NE. PSFWNA) GOTO 47
         WRITE (10, *) 'PS-F-WRONUMARG:  Wrong '
     &           // 'number of arguments '
     &           // 'in call to Procedural'
```

```
      WRITE (10, *) 'Interface (GSR) low-level '
     &          // 'I/O procedure '
     &          // '(source file: PROIOLIB.MAR).'
      WRITE (10, *) 'This error should NEVER '
     &          // 'occur and indicates a '
     &          // 'Procedural Interface (GSR)'
      WRITE (10, *) 'internal validity check.'
      CALL PVAXSP
      GOTO 1000
C     ELSE
   47 IF (ERRCOD .NE. PSFPTL) GOTO 48
      WRITE (10, *) 'PS-F-PROTOOLAR: Prompt '
     &          // 'buffer too large '
     &          // 'error in call to: PSPRCV.'
      WRITE (10, *) 'This error should NEVER '
     &          // 'occur and indicates a '
     &          // 'Procedural Interface (GSR)'
      WRITE (10, *) 'internal validity check.'
      CALL PVAXSP
      GOTO 1000
C     ELSE
C
C     Unknown error message error message.
C
   48 IF (ERRCOD .GE. 512) GOTO 49
      MSSG2 = MSSG1 // 'warning'
      GOTO 51
C     ELSE
   49   IF (ERRCOD .GE. 1024) GOTO 50
        MSSG2 = MSSG1 // 'error '
        GOTO 51
C       ELSE
   50     MSSG2 = MSSG1 // 'fatal error '
C       END IF
C     END IF
   51 WRITE (10, *) MSSG2
      WRITE (10, *) 'code is unrecognized.'
      WRITE (10, *) 'Probable Procedural '
     &          // 'Interface (GSR) Internal '
     &          // 'validity check error.'
C     END IF
 1000 IF ((ERRCOD .LT. PSFPAF) .OR.
     &    (ERRCOD .GT. PSFPPF)) GOTO 2000
      CALL PSFVMSERR ( VMSdef, PIdef )
      WRITE (10, *) 'DEC VAX/VMS Error '
     &          // 'definition is:'
      WRITE (10, *)  VMSdef
      WRITE (10, *) 'Procedural Interface '
     &          // '(GSR) Interpretation of '
     &          // 'DEC VAX/VMS completion code:'
```

```
      WRITE (10, *)  PIdef
      WRITE (10, *) 'DEC VAX/VMS Error code '
     &          // 'value was: ', PsVMSerr ()
C     END IF
 2000 WRITE (10, *)
     RETURN
     END
```

SUBROUTINE PIBMSP

```
C
C      PIBMSP:  Write the "IBM version specific"
C               message to the Error handler file.
C

     WRITE (10, *) 'This error/warning is '
     &         // 'applicable ONLY to the IBM '
     &         // 'version of the'
     WRITE (10, *) 'Procedural Interface (GSR).'
     RETURN
     END
```

SUBROUTINE PVAXSP

```
C
C      PVAXSP:  Write the "DEC VAX/VMS Version
C               specific" message to the Error
C               handler file.
C

     WRITE (10, *) 'This error/warning is '
     &         // 'applicable ONLY to the DEC '
     &         // 'VAX/VMS version of'
     WRITE (10, *) 'the Procedural Interface (GSR).'
     RETURN
     END
```

This is a FORTRAN network creation example.

```
      PROGRAM BlkLevF

      INCLUDE 'PROCONST.FOR/NOLIST'

C
C       Main program:
C
      REAL*4    Deg_rad
      PARAMETER (Deg_rad    = 0.017453292)

      REAL*4    Theta, DTheta, Front (4, 100),
     &          Vecs (4, 100), Zero_vec (3),
     &          Y_Up (3), At (3), From (3), Up (3)
      INTEGER*4 i, k, l, Times
      CHARACTER Name*63, DeviceSpec*1, DeviceName*5,
     &          AttachParameter*80
      LOGICAL*1 PFront (100), PVecs (100)

      CHARACTER Uppercase*1
      EXTERNAL  Err, Uppercase


      DeviceSpec = ' '
      DO WHILE ((DeviceSpec .NE. 'A') .AND.
     &          (DeviceSpec .NE. 'D') .AND.
     &          (DeviceSpec .NE. 'P'))
        WRITE (6, 1) 'Device Interface type = '
     &           // '(Parallel, DMR-11, Asynchronous): _'
        READ  (5, 2)  DeviceSpec
        DeviceSpec = Uppercase (DeviceSpec)
        IF ((DeviceSpec .NE. 'A') .AND.
     &      (DeviceSpec .NE. 'D') .AND.
     &      (DeviceSpec .NE. 'P')) THEN
          WRITE (6, *) 'Invalid device type specified.'
        END IF
      END DO
      DeviceName = ' '
      DO WHILE (DeviceName .EQ. ' ')
        WRITE (6, 1) 'Physical device name (i.e. TT, '
     &           // 'TTA6, XMD0): _'
        READ  (5, 3)  DeviceName
    1   FORMAT (' ', A, $)
    2   FORMAT (1A)
    3   FORMAT (5A)
      END DO
      IF ((DeviceName (2:2)) .EQ. ' ') THEN
        DeviceName (2:) = ':'
      ELSE
```

```
  IF ((DeviceName (3:3)) .EQ. ' ') THEN
    DeviceName (3:) = ':'
  ELSE
    IF ((DeviceName (4:4)) .EQ. ' ') THEN
      DeviceName (4:) = ':'
    ELSE
      DeviceName (5:) = ':'
    END IF
  END IF
 END IF
 IF ((Uppercase (DeviceSpec)) .EQ. 'P') THEN
   AttachParameter = 'Logdevnam=' // DeviceName ·
&              // '/Phydevtyp=PARALLEL'
 ELSE
   IF ((Uppercase (DeviceSpec)) .EQ. 'D') THEN
     AttachParameter = 'Logdevnam=' // DeviceName
&               // '/Phydevtyp=DMR-11'
   ELSE
     AttachParameter = 'Logdevnam=' // DeviceName
&               // '/Phydevtyp=Async'
   END IF
 END IF
 CALL PAttch (AttachParameter, Err)
 At (1) = 0.3
 At (2) = 0
 At (3) = 0
 From (1) = 0
 From (2) = 0
 From (3) = -1
 Up (1) = 0.3
 Up (2) = 1
 Up (3) = 0
 Y_up (1) = 0
 Y_up (2) = 1
 Y_up (3) = 0
 Zero_vec (1) = 0
 Zero_vec (2) = 0
 Zero_vec (3) = 0
 CALL Pinit ( Err )
 CALL Peyebk ( 'eye', 1.0, 0.0, 0.0, 2.0, 0.0,
&          1000.0, 'inten', Err )
 CALL Pseint ( 'inten', .TRUE., 0.5, 1.0,
&          'look', Err )
 CALL PLooka ( 'look', At, From, Up, 'pic', Err )
 CALL Pfn   ( 'atx', 'xvec', Err )
 CALL Pfn   ( 'aty', 'yvec', Err )
 CALL Pfn   ( 'atz', 'zvec', Err )
 CALL Pfn   ( 'fromx', 'xvec', Err )
 CALL Pfn   ( 'fromy', 'yvec', Err )
 CALL Pfn   ( 'fromz', 'zvec', Err )
```

```
CALL Pfn   ( 'ac_at', 'accumulate', Err )
CALL Pfn   ( 'ac_from', 'accumulate', Err )
CALL Pfn   ( 'add_up', 'addc', Err )
CALL PfnN  ( 'sync_up', 'sync', 3, Err )
CALL Pfn   ( 'fix_sync', 'nop', Err )
CALL Pconn ( 'sync_up', 3, 1, 'fix_sync', Err )
CALL Pconn ( 'fix_sync', 1, 3, 'sync_up', Err )
CALL Psnboo ( .TRUE., 3, 'sync_up', Err )
CALL Pfn   ( 'look_at', 'lookat', Err )
CALL Pconn ( 'dials', 1, 1, 'atx', Err )
CALL Pconn ( 'dials', 2, 1, 'aty', Err )
CALL Pconn ( 'dials', 3, 1, 'atz', Err )
CALL Pconn ( 'dials', 5, 1, 'fromx', Err )
CALL Pconn ( 'dials', 6, 1, 'fromy', Err )
CALL Pconn ( 'dials', 7, 1, 'fromz', Err )
CALL Pconn ( 'atx', 1, 1, 'ac_at', Err )
CALL Pconn ( 'aty', 1, 1, 'ac_at', Err )
CALL Pconn ( 'atz', 1, 1, 'ac_at', Err )
CALL Pconn ( 'fromx', 1, 1, 'ac_from', Err )
CALL Pconn ( 'fromy', 1, 1, 'ac_from', Err )
CALL Pconn ( 'fromz', 1, 1, 'ac_from', Err )
CALL Pconn ( 'ac_at', 1, 1, 'sync_up', Err )
CALL Pconn ( 'ac_at', 1, 1, 'add_up',  Err )
CALL Pconn ( 'add_up',1, 2, 'sync_up', Err )
CALL Pconn ( 'sync_up', 1, 1, 'look_at', Err )
CALL Pconn ( 'sync_up', 2, 3, 'look_at', Err )
CALL Pconn ( 'ac_from', 1, 2, 'look_at', Err )
CALL Psnv3d ( At, 2, 'ac_at', Err )
CALL Psnv3d ( From, 2, 'ac_from', Err )
CALL Psnv3d ( Y_up, 2, 'add_up', Err )
CALL Pconn ( 'look_at', 1, 1, 'look', Err )
CALL Pfn   ( 'fix_at', 'const', Err )
CALL Pconn ( 'ac_from', 1, 1, 'fix_at', Err )
CALL Pconn ( 'fix_at', 1, 1, 'ac_at', Err )
CALL Psnv3d ( Zero_vec, 2, 'fix_at', Err )
CALL Psnv3d ( Zero_vec, 1, 'ac_from', Err )
CALL Pinst ( 'pic',"", Err )
Dtheta = 10.0 * Deg_rad
Theta  = -Dtheta
DO i = 1, 36
  Theta = Theta + Dtheta
  CALL Computewave (Theta, Vecs, PVecs)
  DO k=1, 50
    DO l=1, 4
      Front (l, k) = Vecs (l, (k-1)*2+1)
      PFront (k) = PVecs ((k-1) * 2 + 1)
    END DO
  END DO
  CALL Computename ( i, Name )
  CALL Pbegs  ( Name, Err )
```

```
  CALL Pser  ( "", 1, 35, .FALSE., i, "", Err )
  CALL Pifpha ( "", .TRUE., "", Err )
  CALL Pvcbeg ( "", 100, .FALSE., .FALSE., 3,
&           PVsepa, Err )
  CALL Pvclis ( 100, Vecs, PVecs, Err )
  CALL Pvcend ( Err )
  CALL Pvcbeg ( "", 50, .FALSE., .FALSE., 3,
&           PVconn, Err )
  CALL Pvclis ( 50, Front, PFront, Err )
  CALL Pvcend ( Err )
  CALL Pends ( Err )
  CALL Pincl ( Name, 'pic', Err )
 END DO
 CALL Pdisp ( 'eye', Err )
 CALL PSnSt ( 'X', 1, 'Dlabel1', Err )
 CALL PSnSt ( 'Y', 1, 'Dlabel2', Err )
 CALL PSnSt ( 'Z', 1, 'Dlabel3', Err )
 CALL PSnSt ( 'Look At', 1, 'Dlabel4', Err )
 CALL PSnSt ( 'X', 1, 'Dlabel5', Err )
 CALL PSnSt ( 'Y', 1, 'Dlabel6', Err )
 CALL PSnSt ( 'Z', 1, 'Dlabel7', Err )
 CALL PSnSt ( 'From', 1, 'Dlabel8', Err )
 CALL Pdtach ( Err )
 END
```

SUBROUTINE Computename (Nameid, Name)

```
INTEGER*4  NameId
CHARACTER  Name*(*)

INTEGER*4  j, L_name

Name = 'List000"'
L_name = Nameid
j = 7
DO WHILE (L_name .GT. 0)
  Name (j:j) = CHAR (MOD (L_name, 10) + ICHAR ('0'))
  L_name = L_name/10
  j = j - 1
END DO
RETURN
END
```

```
      SUBROUTINE ComputeWave (Theta, VecList, PosLin)

      REAL*4     Theta, VecList (4, 100)
      LOGICAL*1  PosLin (*)

      REAL*4     Amp, Alpha, Beta
      PARAMETER  (Amp   = 0.8, Alpha  = -0.02,
     &          Beta   = 0.2513274123)

      INTEGER*4  i, IAddr

      Iaddr = -1
      DO i = 0, 49
        Iaddr = Iaddr + 2
        Veclist (1, Iaddr) = i / 50.0
        Veclist (2, Iaddr) = Amp * EXP (Alpha * i)
     &                 * cos (Theta - Beta * i)
        Veclist (3, Iaddr) = 0
        Veclist (4, Iaddr) = 1 - i/150.0
        PosLin (  Iaddr) = .TRUE.
        Veclist (1, Iaddr+1) = Veclist (1, Iaddr)
        Veclist (2, Iaddr+1) = 0
        Veclist (3, Iaddr+1) = 0.5
        Veclist (4, Iaddr+1) = Veclist (4, Iaddr)
        PosLin (  Iaddr+1) = .TRUE.
      END DO
      RETURN
      END


      CHARACTER*1 FUNCTION Uppercase (Chara)
      CHARACTER   Chara*(*)
      IF (((Chara (1:1)) .GE. 'a') .AND.
     &    ((Chara (1:1)) .LE. 'z')) THEN
        Uppercase = CHAR (ICHAR (Chara (1:1)) - 32)
      ELSE
        Uppercase = Chara
      END IF
      RETURN
      END


C
C
C
C  The following Error Handler demonstrates the general
C  overall recommended form that the user's own error
C  handler should follow.
```

```
C
C  This error handler upon being invoked writes ALL
C  messages to the data file: 'PROERROR.LOG'. Error
C  and warning explanation messages are written to
C  a data file for 2 reasons:
C
C
C     1. The error handler should NOT immediately
C        write information out on the PS 300 screen
C        since the explanatory text defining the error
C        or warning condition may be taken as data by
C        the PS 300 and therefore wind up not being
C        displayed on the PS 300 screen (as in the
C        case of a catastrophic data transmission
C        error).
C
C     2. The logging of errors and warnings to a
C        logfile allows any errors and/or warnings
C        to be reviewed at a later time.
C
C
C
```

```
      SUBROUTINE ERR ( ERRCOD )

C
C     Procedural Interface (GSR) error handler:
C

      INCLUDE  'PROCONST.FOR/NOLIST'
      INTEGER*4  ERRCOD
      INTEGER*4  PsVMSerr
      LOGICAL    FILOPN
      DATA       FILOPN /.FALSE./
      EXTERNAL   PsVMSerr, DETERH, PIDCOD

      IF (FILOPN) GOTO 1
C
C     Open error file for logging of errors:
C
        OPEN (UNIT=10, FILE='PROERROR.LOG', STATUS='NEW',
     &       DISP='KEEP', ORGANIZATION='SEQUENTIAL',
     &       ACCESS='SEQUENTIAL', CARRIAGECONTROL='LIST')
        FILOPN = .TRUE.
C     END IF
    1 CALL PIDCOD (ERRCOD)
      IF (ERRCOD .LT. 512) GOTO 3
```

```
      WRITE (10, *) 'PS-I-ATDCOMLNK:  Attempting to '
     &          // 'detach PS 300/Host communications '
     &          // 'link.'
C
C       When we attempt to perform the Detach, use a
C       different error handler so as not to get caught
C       in a recursive loop if we consistently get an
C       error when attempting to detach.
C
        CALL PDTACH (DETERH)
        CLOSE (UNIT=10)
        IF ((ERRCOD .LT. PSFPAF) .OR.
     &      (ERRCOD .GT. PSFPPF)) GOTO 2
C
C         Identify VMS error if there was one
C
         CALL LIB$STOP (%VAL (PsVMSerr ()))
         GOTO 3
C       ELSE
    2    STOP
C       END IF
C     END IF
    3 RETURN
      END


      SUBROUTINE DETERH (ERRCOD)

C
C       Main Error handler Detach error handler:
C

      INTEGER*4  ERRCOD
      EXTERNAL   PIDCOD

      WRITE (10, *) 'PS-I-ERRWARDET:  Error/warning '
     &          // 'trying to Detach '
     &          // 'the communications'
      WRITE (10, *) 'link between the PS 300 and the host.'
      CALL PIDCOD (ERRCOD)
      RETURN
      END
```

SUBROUTINE PIDCOD (ERRCOD)

```
C
C     PIDCOD: Identify Procedural Interface (GSR) Completion
C          code.
C

      INCLUDE  'PROCONST.FOR/NOLIST'
      INTEGER*4  ERRCOD
      CHARACTER  VMSDEF*133, PIDEF*133
      INTEGER*4  PsVMSerr
      CHARACTER  MSSG1*55, MSSG2*67
      PARAMETER  (MSSG1 = 'PS-W-UNRCOMCOD:  Procedural '
     &            // 'Interface '
     &            // '(GSR) completion ')
      EXTERNAL   PsVMSerr

      WRITE (10, *) 'PS-I-PROERRWAR:  Procedural '
     &        // 'Interface warning/'
     &        // 'error completion code was '
      WRITE (10, *) 'received.'
      IF (ERRCOD .NE. PSWBNC) GOTO 1
        WRITE (10, *) 'PS-W-BADNAMCHR:  Bad character '
     &          // 'in name was '
     &          // 'translated to:  "_".'
        GOTO 1000
C     ELSE
    1 IF (ERRCOD .NE. PSWNTL) GOTO 2
        WRITE (10, *) 'PS-W-NAMTOOLON:  Name too '
     &          // 'long.  Name was '
     &          // 'truncated to '
        WRITE (10, *) '256 characters.'
        GOTO 1000
C     ELSE
    2 IF (ERRCOD .NE. PSWSTL) GOTO 7
        WRITE (10, *) 'PS-W-STRTOOLON:  String too '
     &          // 'long.  String '
     &          // 'was truncated '
        WRITE (10, *) 'to 240 characters.'
        GOTO 1000
C     ELSE
    7 IF (ERRCOD .NE. PSWAAD) GOTO 8
        WRITE (10, *) 'PS-W-ATTALRDON:  Attach '
     &          // 'already done.  '
     &          // 'Multiple call to PAttch without'
        WRITE (10, *) 'intervening PDtach call ignored.'
        GOTO 1000
```

```
C     ELSE
    8 IF (ERRCOD .NE. PSWAKS) GOTO 9
        WRITE (10, *) 'PS-W-ATNKEYSEE:  Attention key '
     &          // 'seen (depressed).'
        CALL PIBMSP
        GOTO 1000
C     ELSE
    9 IF (ERRCOD .NE. PSWBGC) GOTO 10
        WRITE (10, *) 'PS-W-BADGENCHR:  Bad generic '
     &          // 'channel character.  Bad '
        WRITE (10, *) 'character in string sent via:  '
     &          // 'PPutGX  was translated to '
        WRITE (10, *) 'a blank.'
        CALL PIBMSP
        GOTO 1000
C     ELSE
   10 IF (ERRCOD .NE. PSWBSC) GOTO 11
        WRITE (10, *) 'PS-W-BADSTRCHR:  Bad '
     &          // 'character in string was '
     &          // 'translated to a blank.'
        CALL PIBMSP
        GOTO 1000
C     ELSE
   11 IF (ERRCOD .NE. PSWBPC) GOTO 12
        WRITE (10, *) 'PS-W-BADPARCHR:  Bad parser '
     &          // 'channel character.  Bad '
     &          // 'character in string sent to'
        WRITE (10, *) 'PS 300 parser via:  PPutP  '
     &          // 'was translated to a blank.'
        CALL PIBMSP
        GOTO 1000
C     ELSE
   12 IF (ERRCOD .NE. PSEIMC) GOTO 13
        WRITE (10, *) 'PS-E-INVMUXCHA:  Invalid '
     &          // 'multiplexing channel '
     &          // 'specified in call to:'
        WRITE (10, *) 'PMuxCI, PMuxP, or PMuxG.'
        GOTO 1000
C     ELSE
   13 IF (ERRCOD .NE. PSEIVC) GOTO 14
        WRITE (10, *) 'PS-E-INVVECCLA:  Invalid '
     &          // 'vector list class '
     &          // 'specified'
        WRITE (10, *) 'in call to:  PVcBeg.'
        GOTO 1000
C     ELSE
   14 IF (ERRCOD .NE. PSEIVD) GOTO 15
        WRITE (10, *) 'PS-E-INVVECDIM:  Invalid '
     &          // 'vector list dimension '
     &          // 'specified in call to'
```

```
      WRITE (10, *) 'PVcBeg.'
      GOTO 1000
C     ELSE
   15 IF (ERRCOD .NE. PSEPOE) GOTO 16
      WRITE (10, *) 'PS-E-PREOPEEXP: Prefix '
     &          // 'operator call was '
     &          // 'expected.'
      GOTO 1000
C     ELSE
   16 IF (ERRCOD .NE. PSEFOE) GOTO 17
      WRITE (10, *) 'PS-E-FOLOPEEXP: Follow '
     &          // 'operator call was '
     &          // 'expected.'
      GOTO 1000
C     ELSE
   17 IF (ERRCOD .NE. PSELBE) GOTO 18
      WRITE (10, *) 'PS-E-LABBLKEXP: Call to '
     &          // 'PLaAdd or PLaEnd was '
     &          // 'expected.'
      GOTO 1000
C     ELSE
   18 IF (ERRCOD .NE. PSEVLE) GOTO 19
      WRITE (10, *) 'PS-E-VECLISEXP: Call to '
     &          // 'PVcLis or PVcEnd '
     &          // 'was expected.'
      GOTO 1000
C     ELSE
   19 IF (ERRCOD .NE. PSEAMV) GOTO 20
      WRITE (10, *) 'PS-E-ATTMULVEC: Attempted '
     &          // 'multiple call '
     &          // 'sequence to PVcLis is NOT'
      WRITE (10, *) 'permitted for BLOCK '
     &          // 'normalized vectors.'
      GOTO 1000
C     ELSE
   20 IF (ERRCOD .NE. PSEMLB) GOTO 21
      WRITE (10, *) 'PS-E-MISLABBEG: Missing '
     &          // 'label block begin call. '
     &          // 'Call to PLaAdd or PLaEnd'
      WRITE (10, *) 'without call to: PLaBeg.'
      GOTO 1000
C     ELSE
   21 IF (ERRCOD .NE. PSEMVB) GOTO 22
      WRITE (10, *) 'PS-E-MISVECBEG: Missing '
     &          // 'vector list begin '
     &          // 'call. Call to PVcLis'
      WRITE (10, *) 'or PVcEnd without call '
     &          // 'to: PVcBeg.'
      GOTO 1000
```

```
C     ELSE
   22 IF (ERRCOD .NE. PSENUN) GOTO 23
        WRITE (10, *) 'PS-E-NULNAM:  Null name '
     &          // 'parameter is not allowed.'
        GOTO 1000
C     ELSE
   23 IF (ERRCOD .NE. PSEBCT) GOTO 24
        WRITE (10, *) 'PS-E-BADCOMTYP:  Bad '
     &          // 'comparison type operator '
     &          // 'specified in '
        WRITE (10, *) 'call to:  PIfLev.'
        GOTO 1000
C     ELSE
   24 IF (ERRCOD .NE. PSEIFN) GOTO 25
        WRITE (10, *) 'PS-E-INVFUNNAM:  Invalid '
     &          // 'function name.  '
     &          // 'Attempted PS 300'
        WRITE (10, *) 'function instance failed '
     &          // 'because the named '
     &          // 'function cannot possibly'
        WRITE (10, *) 'exist.  The function name '
     &          // 'identifying the '
     &          // 'function type to instance'
        WRITE (10, *) 'was longer than 256 characters.'
        GOTO 1000
C     ELSE
   25 IF (ERRCOD .NE. PSENNR) GOTO 26
        WRITE (10, *) 'PS-E-NULNAMREQ:  Null name '
     &          // 'parameter is '
     &          // 'required in operate node'
        WRITE (10, *) 'call following a PPref or '
     &          // 'PFoll procedure call.'
        GOTO 1000
C     ELSE
   26 IF (ERRCOD .NE. PSETME) GOTO 27
        WRITE (10, *) 'PS-E-TOOMANEND:  Too '
     &          // 'many END_STRUCTURE calls '
     &          // 'invoked.'
        GOTO 1000
C     ELSE
   27 IF (ERRCOD .NE. PSENOA) GOTO 28
        WRITE (10, *) 'PS-E-NOTATT:  The PS 300 '
     &          // 'communications link '
     &          // 'has not '
        WRITE (10, *) 'yet been established.  '
     &          // 'PAttch has not been '
     &          // 'called or failed.'
        GOTO 1000
```

```
C     ELSE
   28 IF (ERRCOD .NE. PSEODR) GOTO 38
        WRITE (10, *) 'PS-E-OVEDURREA: An '
     &          // 'overrun occurred during '
     &          // 'a read operation.'
        WRITE (10, *) 'The specified input buffer '
     &          // 'in call to: PGET  '
     &          // 'or: PGETW'
        WRITE (10, *) 'was too small and '
     &          // 'truncation has occurred.'
        GOTO 1000
C     ELSE
   38 IF (ERRCOD .NE. PSEPDT) GOTO 39
        WRITE (10, *) 'PS-E-PHYDEVTYP: Missing '
     &          // 'or invalid physical '
     &          // 'device type'
        WRITE (10, *) 'specifier in call to PAttch.'
        CALL PVAXSP
        GOTO 1000
C     ELSE
   39 IF (ERRCOD .NE. PSELDN) GOTO 40
        WRITE (10, *) 'PS-E-LOGDEVNAM: Missing '
     &          // 'or invalid logical '
     &          // 'device name'
        WRITE (10, *) 'specifier in call to PAttch.'
        CALL PVAXSP
        GOTO 1000
C     ELSE
   40 IF (ERRCOD .NE. PSEADE) GOTO 41
        WRITE (10, *) 'PS-E-ATTDELEXP: Attach '
     &          // 'parameter string '
     &          // 'delimiter'
        WRITE (10, *) '"/" was expected.'
        CALL PVAXSP
        GOTO 1000
C     ELSE
   41 IF (ERRCOD .NE. PSFPAF) GOTO 42
        WRITE (10, *) 'PS-F-PHYATTFAI: '
     &          // 'Physical attach operation '
     &          // 'failed.'
        GOTO 1000
C     ELSE
   42 IF (ERRCOD .NE. PSFPDF) GOTO 43
        WRITE (10, *) 'PS-F-PHYDETFAI: Physical '
     &          // 'detach operation '
     &          // 'failed.'
        GOTO 1000
```

```
C     ELSE
   43 IF (ERRCOD .NE. PSFPGF) GOTO 44
         WRITE (10, *) 'PS-F-PHYGETFAI: Physical '
     &         // 'GET operation failed.'
         GOTO 1000
C     ELSE
   44 IF (ERRCOD .NE. PSFPPF) GOTO 45
         WRITE (10, *) 'PS-F-PHYPUTFAI: Physical '
     &         // 'PUT operation failed.'
         GOTO 1000
C     ELSE
   45 IF (ERRCOD .NE. PSFBTL) GOTO 46
         WRITE (10, *) 'PS-F-BUFTOOLAR: Buffer '
     &         // 'too large error in '
     &         // 'call to: PSPUT.'
         WRITE (10, *) 'This error should NEVER '
     &         // 'occur and indicates a '
     &         // 'Procedural Interface (GSR)'
         WRITE (10, *) 'internal validity check.'
         CALL PVAXSP
         GOTO 1000
C     ELSE
   46 IF (ERRCOD .NE. PSFWNA) GOTO 47
         WRITE (10, *) 'PS-F-WRONUMARG: Wrong '
     &         // 'number of arguments '
     &         // 'in call to Procedural'
         WRITE (10, *) 'Interface (GSR) low-level '
     &         // 'I/O procedure '
     &         // '(source file: PROIOLIB.MAR).'
         WRITE (10, *) 'This error should NEVER '
     &         // 'occur and indicates a '
     &         // 'Procedural Interface (GSR)'
         WRITE (10, *) 'internal validity check.'
         CALL PVAXSP
         GOTO 1000
C     ELSE
   47 IF (ERRCOD .NE. PSFPTL) GOTO 48
         WRITE (10, *) 'PS-F-PROTOOLAR: Prompt '
     &         // 'buffer too large '
     &         // 'error in call to: PSPRCV.'
         WRITE (10, *) 'This error should NEVER '
     &         // 'occur and indicates a '
     &         // 'Procedural Interface (GSR)'
         WRITE (10, *) 'internal validity check.'
         CALL PVAXSP
         GOTO 1000
C     ELSE
C
C     Unknown error message error message.
```

```
C
  48 IF (ERRCOD .GE. 512) GOTO 49
     MSSG2 = MSSG1 // 'warning'
     GOTO 51
C    ELSE
  49   IF (ERRCOD .GE. 1024) GOTO 50
       MSSG2 = MSSG1 // 'error '
       GOTO 51
C      ELSE
  50     MSSG2 = MSSG1 // 'fatal error '
C      END IF
C    END IF
  51 WRITE (10, *) MSSG2
     WRITE (10, *) 'code is unrecognized.'
     WRITE (10, *) 'Probable Procedural '
    &         // 'Interface (GSR) Internal '
    &         // 'validity check error.'
C    END IF
 1000 IF ((ERRCOD .LT. PSFPAF) .OR.
    &   (ERRCOD .GT. PSFPPF)) GOTO 2000
      CALL PSFVMSERR ( VMSdef, PIdef )
      WRITE (10, *) 'DEC VAX/VMS Error '
    &          // 'definition is:'
      WRITE (10, *)  VMSdef
      WRITE (10, *) 'Procedural Interface '
    &          // '(GSR) Interpretation of '
    &          // 'DEC VAX/VMS completion code:'
      WRITE (10, *)  PIdef
      WRITE (10, *) 'DEC VAX/VMS Error code '
    &          // 'value was: ', PsVMSerr ()
C    END IF
 2000 WRITE (10, *)
     RETURN
     END
```

SUBROUTINE PIBMSP

```
C
C      PIBMSP:  Write the "IBM version specific"
C               message to the Error handler file.
C

     WRITE (10, *) 'This error/warning is '
    &         // 'applicable ONLY to the IBM '
    &         // 'version of the'
     WRITE (10, *) 'Procedural Interface (GSR).'
     RETURN
     END
```

```
      SUBROUTINE PVAXSP

C
C     PVAXSP:  Write the "DEC VAX/VMS Version
C              specific" message to the Error
C              handler file.

C
      WRITE (10, *) 'This error/warning is '
     &        // 'applicable ONLY to the DEC '
     &        // 'VAX/VMS version of'
      WRITE (10, *) 'the Procedural Interface (GSR).'
      RETURN
      END
```

APPENDIX B. HOST_MESSAGE

This appendix contains the function network diagram and functional description of HOST_MESSAGE (an instance of the intrinsic function HOLD_MESSAGE) that supports the subroutines PGETW and PGET of the GSRs. This function is already part of the PS 300 system. When using the GSRs, all messages sent from the PS 300 to the host must be sent via this function.

The function HOST_MESSAGE is a F:NOP function directly connected to the function HOST_MESSAGEB. It is recommended that the user always send PS 300 output destined for the host computer to HOST_MESSAGE rather than HOST_MESSAGEB since the name of the latter function may change with a future release of runtime software.

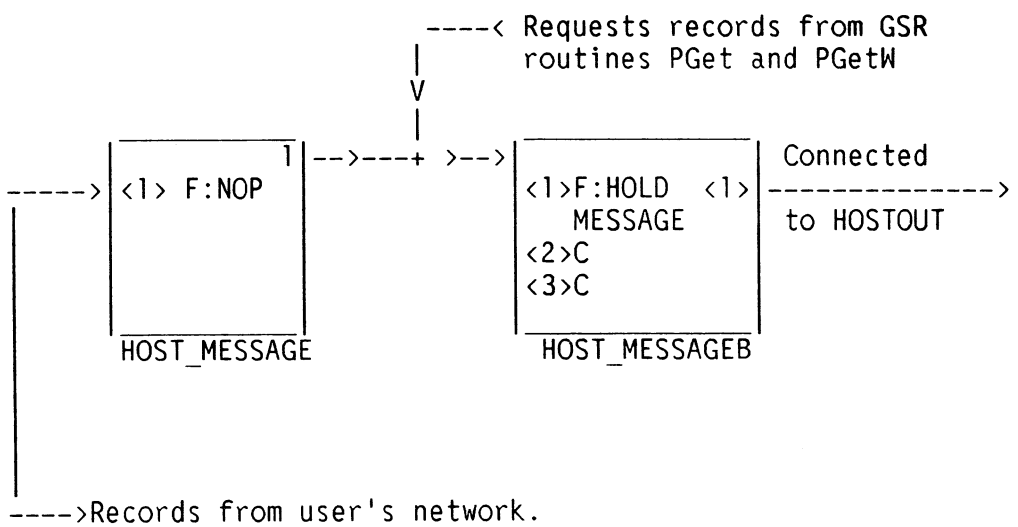

Figure B-1. Hold_Message Function Network Diagram

HOLD_MESSAGE:

INPUTS:

<1>: Qpackets of messages to be sent to the Host and Qintegers used to trigger the messages as follows:

FIX(0): Clear any messages waiting the FIFO queue of messages to be sent to the Host.

FIX(1): If a message is waiting, send it. Otherwise send the message indicating: "No-messages" as determined by input <3>.

FIX(2): If a message is waiting, send it. Otherwise, wait until a Qpacket message arrives on input <1> and then immediately send the message.

<2>C: Message Terminator Qpacket that is added to the end of messages arriving on input <1> just prior to transmission to the host.

The default input value for input <2> is a carriage return: CHR (13).

<3>C: "No-messages" Qpacket. If this function receives a FIX (1) on input <1>, then the message on this constant queue is sent ONLY if there are no other messages waiting to be sent on input <1>. Otherwise, the first message on the FIFO queue of messages is sent from output <1> with the Message Terminator Qpacket as defined by input <2>.

The default input value for input <3> is a carriage return: CHR (13).

OUTPUTS:

<1>: Qpacket sent to the Host Computer in response to the receipt of either a FIX (1) or FIX (2) on input <1>.

The GSR subroutines: PGet and PGetW specifically interrogate the function: HOST_MESSAGEB for input back to the host.

The subroutine PGet is used to "poll" the PS 300 for data. If a message exists on the FIFO queue of HOST_MESSAGEB, then that message is removed from the queue and is returned by PGet. If no message was present in the input queue of HOST_MESSAGEB then the special: "No-messages" message as defined by input <3> of HOST_MESSAGE is returned.

The subroutine PGetW is similar in functionality to PGet with one important difference. PGetW will NOT return to the caller until a message has been received from the PS 300. If no messages are present on the input queue of HOST_MESSAGEB, then the caller of PGetW (Get message and wait for completion) will wait until a message is sent to input <1> of HOST_MESSAGEB.

NOTE

Messages received from the PS 300 via PGet and PGetW may need to be "trimmed" of the trailing character(s) as defined by inputs <2> and <3> of HOST_MESSAGEB if either of them is changed from the default value of carriage return (Character 13). The DEC VAX/VMS FORTRAN GSR will remove a single trailing carriage return from the message. Thus if a poll operation is requested and no messages are present, the GSR returns a zero-length message to the caller indicating that no messages were present because the default "No-message" message on input <3> of HOST_MESSAGEB is a carriage return. Similary, calls to PGetW return the proper length. However, if the user chooses to change the HOST_MESSAGEB inputs <2> or <3>, then the user must compensate for any side effects so produced when calling PGet or PGetW.

USER'S MANUAL FOR PS 300 HOST-RESIDENT I/O SUBROUTINES

Supports DEC VAX/VMS and DEC PDP-11/RSX-11M Systems

Supported Under Software Version A1

CONTENTS

INTRODUCTION

The PS 300 Host-Resident I/O Subroutines (PSIOs) provide the user with a standard method of communication between the PS 300 and the host system. The subroutines are distributed on magtape to each customer. These subroutines must be loaded, linked, and compiled on the host system by the user. The package is supported under Digital Equipment Corporation VAX/VMS (Version 2.3) and RSX-11M (Version 3.2) operating systems.

The user of this package should be familiar with FORTRAN, the PS 300 command language, and the command language for VMS and/or RSX-11M operating systems.

The PSIO package is supported under FORTRAN IV PLUS (FFP) and FORTRAN IV.

The PSIO package described in this manual is supported only under PS 300 Graphic Software Version P5 and higher.

This manual is intended for users who are fully acquainted with the PS 300 graphic system conventions and commands. No attempt has been made to define the terms used in this manual.

There are two levels of routines in the subroutines. The first level contains the subroutines PSETUP, PSEXIT, PSREAD, PSPOLL, PSVECS, PSSEND, PSCHAR, and PSFIXI. These routines will run on machines that support Logical *1 variables.

The second level of subroutines is written for the particular machine that the I/O subroutines will run on. These routines are written in FORTRAN and assembly language to get the maximum possible throughput. The routines at the second level will support input and output to more than one device, such as a terminal or DMR11 driver, that can be selected at execution time.

To install the PSIOs, refer to Volume 5 of this document set. Volume 5 also contains data communication information for those users wishing to write their own I/O subroutine package.

Appendix A contains a functional description of HOLD_MESSAGE that supports the subroutine PSREAD and PSPOLL.

Appendix B lists reserved words that are used by E&S to name subroutines, COMMON blocks, functions, or BLOCK DATA in the PS 300 Host-Resident Subroutines.

1.1 DESCRIPTION OF THE HOST-RESIDENT I/O SUBROUTINES

This section describes the top-level subroutines. These subroutines are written for transportability between host systems. Section 1.2 describes the second level subroutines that are written specifically for different operating systems. The top level subroutines and their arguments are:

- PSETUP (IN, IOUT, INIT, LEN, IARRAY)
- PSEXIT (LEN, IARRAY)
- PSREAD (INLEN, INBUF, NUMBYT)
- PSPOLL (INLEN, INBUF, NUMBYT)
- PSVECS (IVT, IVC, VECS, IPL, DELIM, IOSTAT)
- PSSEND (LEN, IOUTBF)
- PSCHAR (LEN, IOUTBF, FLUSH)
- PSFIXI (DFTINI)

1.1.1 PSETUP

The PSETUP subroutine sets up the communications link to the PS 300. If the "INIT" argument is true, a reset message is sent to the PS 300. The routine initializes all LEVEL 1 COMMON BLOCKS.

FORTRAN Calling Sequence:

```
CALL PSETUP (IN, IOUT, INIT, LEN, IARRAY)
```

Where:

- **IN** (INTEGER) is the input logical unit number used for the asynchronous line under RSX-11M. (Machine dependent for systems other than RSX-11M and VAX.)
- **IOUT** (INTEGER) is the output logical unit number for the DMR11 line under RSX-11M. (Machine dependent for systems other than RSX-11M and VAX.)
- **INIT** (LOGICAL) indicates whether or not to initialize the PS 300. A .TRUE. indicates that the command is to be issued and a .FALSE. indicates that no command is to be issued. If this parameter has other than a logical value, .FALSE. is assumed.
- **LEN** (INTEGER) is an integer (>=2) that specifies the length of the array.
- **IARRAY** (INTEGER)

  PASSED - IARRAY(2) specifies the input/output device:
  0 = Low speed device
  1 = High Speed Device
  3 ..n (Machine Dependent)

  RETURNED - IARRAY(1) = Error Status
  No error (0) is always returned.

Example

```
I=7
IO=7
INIT=.TRUE.
L=2
IARRAY (2)=1
CALL PSETUP (I,IO,INIT,L,IARRAY)
```

1.1.2 PSEXIT

The PSEXIT subroutine disconnects the communication link to the PS 300. This subroutine does not issue any commands to the PS 300. Therefore, when PSEXIT is called, the PS 300 is left "as is".

FORTRAN Calling Sequence:

```
CALL PSEXIT (LEN, IARRAY)
```

Where:

- **LEN** (INTEGER) is an integer (>=1) that specifies the length of the array.
- **IARRAY** (INTEGER)

  Returned - IARRAY(1) = Error Status.

  No error (0) is always returned.

Examples

```
CALL PSEXIT (1,IARRAY)
```

or

```
L=1
CALL PSEXIT (L,IARRAY)
```

1.1.3 PSREAD

The PSREAD subroutine reads a character buffer from the PS 300 up to 256 bytes in length from the PS 300 via HOST_MESSAGE. The subroutine will wait until a message is returned from the PS 300 before it returns to the calling program.

PSREAD will only return one non-null message when called. If more than one message is in HOST_MESSAGE's input queue, PSREAD must be called repeatedly until all messages have been processed.

FORTRAN Calling Sequence:

```
CALL PSREAD (INLEN, INBUF, NUMBYT)
```

Where:

- **INLEN** (INTEGER) specifies the size of the input array in bytes.
- **INBUF** (LOGICAL*1) is the input buffer for the record that is read from the PS 300. The input_array can be any data type (except CHARACTER) but must be accessed as if it were EQUIVALENCEd to a LOGICAL*1 array or BYTE array of length input_array_length.
- **NUMBYT** (INTEGER) returns the actual number of characters in INBUF.

Example

```
CALL PSREAD (72, INBUF, NUMBYT)
```

or

```
I=72
CALL PSREAD (I, INBUF, NUMBYT)
```

1.1.4 PSPOLL

The PSPOLL subroutine requests a character buffer up to 256 bytes long from the PS 300, via HOST_MESSAGE, if and only if a record is available at the time the subroutine is called. If there is no message waiting in the PS 300 to be sent to the host, PSPOLL returns control to the calling program with a buffer count of zero. PSPOLL is used to poll the PS 300 for input records.

PSPOLL will only return one null message or one non-null message when called. If more than one message is in HOST_MESSAGES's input queue, PSPOLL must be called again to obtain the next message. PSPOLL will return with a null message when all messages have been read.

FORTRAN Calling Sequence:

```
CALL PSPOLL (INLEN, INBUF, NUMBYT)
```

Where:

- **INLEN** (INTEGER) specifies the size of the input array in bytes.
- **INBUF** (LOGICAL *1) is the input buffer for the record that is read from the PS 300. INBUF can be any type (except CHARACTER) but must be accessed as if it were EQUIVALENCEd to a LOGICAL*1 array or BYTE array of length input_array_length.
- **NUMBYT** (INTEGER) returns the actual number of characters in INBUF.

Example

```
CALL PSPOLL (72, INBUF, NUMBYT)
```

or

```
I=72
CALL PSPOLL (I, INBUF, NUMBYT)
```

1.1.5 PSVECS

The PSVECS subroutine sends a vector list to the PS 300. On the first call to this routine, it will send either the character message 'V2D' or 'V3D' to the PS 300 for the respective vector type. PSVECS allows an array of single-precision real numbers to be output without the user program having to ENCODE them as character strings. All numbers sent to the PS 300 are in a compact 8-bit binary form.

If a vector list is to be block normalized, only one call to PSVECS is allowed to define the entire list. Multiple calls to PSVECS are allowed to define a given vector-normalized vector list. The restriction to a single call is not imposed on the vector-normalized vector list because the normalization occurs on a per vector basis, rather than on a set of vectors.

Once the block-normalized vectors have been sent to the PS 300, no mechanism is currently available to renormalize them, should subsequent calls to PSVECS require it.

If the vector count passed in a call to PSVECS is larger than the actual vector count, a fatal access violation error will occur.

The vector list command sent (via PSSEND) to the PS 300 prior to the initial PSVECS call must agree in type to the vector type parameter passed to PSVECS.

The vector list command and PSVECS-supported vector list options must be sent to the PS 300 in a prior call to PSSEND. The scope of PSVECS-supported vector list options is as follows:

The subroutine PSFIXI is used to change the default vector intensity used in PSVECS.

Supported Vector List Options

- BLOCK NORMALIZED (Default is vector normalized.)
- @DOTS or ITEMIZED (No default. Vector list command must explicitly state DOTS or ITEMIZED.)
- Implicit Z = 0 for 2D vector lists.

Unsupported Vector List Options

- WITH PATTERN (Therefore, the user program must generate all coordinate values for a patterned vector list.)

- CONNECTED or SEPARATE (Therefore, the user program must set up the position/line array such that the vectors are itemized as follows:

  CONNECTED - P,L,L,...L
  SEPARATE - P,L,P,L,...P,L

- Y = y Z = z DY = delta y DZ = delta z
  (Therefore, the user program must generate all coordinate values for constant or linearly varying y and z.)

- INTERNAL UNITS are not necessary, as the binary format is in internal units.

FORTRAN Calling Sequence:

CALL PSVECS (IVT, IVC, VECS, IPL, DELIM, IOSTAT)

Where:

- **IVT** (INTEGER) specifies the vector type:

  1) vector normalized 2D with specified intensities
  2) vector normalized 3D with specified intensities
  3) vector normalized 2D with default intensity
  4) vector normalized 3D with default intensity
  5) block normalized 2D with default intensity
  6) block normalized 3D with default intensity

These types are illustrated in Table 1-1. If this parameter has a value outside the defined range of vec_types, a fatal error occurs.

- **IVC** (INTEGER) specifies the count of vectors in VECS and count of logical values in the IPL. If the count is zero or negative, no vector data are sent to the PS 300. (See DELIM.)

- **VECS** (REAL) is an array of single-precision real numbers representing the vector coordinates and [intensities]. PSVECS accesses VECS as though it was one dimensional. Therefore, if the user program defines VECS as multi-dimensional, it will be accessed by column.

- **IPL** (LOGICAL) is an array of logical values (.TRUE. or .FALSE.) that itemizes the associated vector in the vec_array as a position vector (.FALSE.) or line vector (.TRUE.). There must be a logical value for each vector; otherwise, the routine assumes it is a position vector (.FALSE.).

- **DELIM** (LOGICAL) indicates whether the vector list will be delimited after the current group of vectors is sent. A .TRUE. indicates that the vector list is to be delimited and a .FALSE. indicates that the vector list is not to be delimited. The latter case allows multiple calls to PSVECS to define a vector list. If this parameter has other than a logical value, .FALSE. is assumed.

- **IOSTAT** (INTEGER) is used to report error status.

  Returned - Error Status

  No error (0) is always returned.

Example

```
CALL PSVECS (4, 50, VECS, IPL, .TRUE.,IOSTAT)
```

or

```
IVT=4
IVC=50
DELIM=.TRUE.
CALL PSVECS (IVT, IVC, VECS, IPL, DELIM, IOSTAT)
```

Table 1-1. Types of Vectors in IVT

vec-type number	Description of vec-array contents	Size of vec-array	Size of position line-array
1* X,Y	and intrinsic intensity specified	vec-count*3	vec-count
2* X,Y,Z	and intrinsic intensity specified	vec-count*4	vec-count
3* X,Y	default intensity	vec-count*2	vec-count
4* X,Y,Z	default intensity	vec-count*3	vec-count
5** X,Y	default intensity	vec-count*2	vec-count
6** X,Y,Z	default intensity	vec-count*3	vec-count

* Vec-types numbers 1, 2, 3, and 4 are vector normalized.
** Vec-types numbers 5 and 6 are block normalized.

Default intensity is initially 1.0 or determined by PSFIXI.

CAUTION

Once transmission of binary data has started, the complete vector list must be transmitted. If the program should fail in the middle of a vector list transmission, the PS 300 will be left in an unknown state and must be rebooted.

1.1.6 PSSEND

The PSSEND subroutine sends a character buffer up to 256 bytes in length to the PS 300. The buffer is sent immediately.

FORTRAN Calling Sequence:

```
CALL PSSEND (LEN, IOUTBF)
```

Where:

- **LEN** (INTEGER) specifies the number of bytes in the number of characters to send. If the byte_count is zero or negative no bytes are written to the PS 300.
- **IOUTBF** (LOGICAL *1) is the array of characters to send. IOUTBF can be any data type except character, but must be accessed as if it were EQUIVALENCEd to a LOGICAL*1 array or BYTE array.

Example

```
CALL PSSEND (20, IOUTBF)
```

or

```
ICNT=20
CALL PSSEND (ICNT, IOUTBF)
```

Please Note: In calls to PSSEND and PSCHAR, lines of text that do not end in a ";", should be terminated with a ' ' (space).

1.1.7 PSCHAR

The PSCHAR subroutine sends a character buffer to the PS 300. The message is packed with the previous PSCHAR messages that had the FLUSH flag set to FALSE before being sent to the PS 300. When the FLUSH flag is set to TRUE, this message and any previous messages it might contain are sent to the PS 300.

FORTRAN Calling Sequence:

```
CALL PSCHAR (LEN, IOUTBF, FLUSH)
```

Where:

- **LEN** (INTEGER) is the number of characters in the buffer to be sent.
- **IOUTBF** (LOGICAL) is the array of characters to send.
- **FLUSH** (LOGICAL) is the flag to send or not send this message. When FLUSH is .TRUE., this message and any previous messages (that had their flag set to FALSE) are sent immediately to the PS 300. When FLUSH IS .FALSE., the message is not sent, but is placed in the buffer and waits until another call to PSCHAR is made with the flag set to TRUE or when the buffer is full.

Example

```
CALL PSCHAR (1, IOUTBF,.TRUE.)
```

or

```
LEN=1
FLUSH=.TRUE.
CALL PSCHAR (LEN, IOUTBF, FLUSH)
```

1.1.8 PSFIXI

The subroutine PSFIXI changes the default vector intensity used by the Subroutine PSVECS at program execution time Initial value is 1.0. (full intensity).

FORTRAN Calling Sequence:

```
CALL PSFIXI (DFTINI)
```

Where:

- **DFTINI** (REAL) is a real number between 0.0 and 1.0 that specifies the maximum intensity.

Example

```
CALL PSFIXI (1.0)
```

or

```
DFTINI=1.0
CALL PSFIXI (DFTINI)
```

1.2 SECOND LEVEL SUBROUTINES

These secondary subroutines are designed for specific systems. The definitions below generally outline what the subroutines do. A more specific description of each routine follows.

A description of the escape and count mode referred to below and used by the PS 300 for host communication is described in Volume 5, **System Manager Reference.**

VAX Set - These subroutines send the data to the PS 300 over an asynchronous serial line in 8-bit character count mode and over the DMR11 in 8-bit form. The error subroutine generates error messages in trace back form.

RSX Set - These subroutines send the data to the PS 300 over an asynchronous serial line in 8-bit character count mode and over the DMR11 in 8-bit form. The error subroutine generates error messages in trace back form.

Brief Description of 8-bit Count Mode Format for Asychronous Line

8-Bit Count Mode Format:

Byte 1 Transmitted - Frame Start Character
Byte 2 Transmitted - Most Significant Bits of 16-bit message count.
Byte 3 Transmitted - Least Significant Bits of 16-bit message count.
Bytes 4 through n - 8-bit data.

The first 3 bytes are not included in message count. Byte 4 is actually the muxing character "1".

1.2.1 PSCON

This subroutine establishes the connection between the PS 300 and host I/O device. After the connection to the PS 300 has been made, the subroutine calls PSPKWM which pokes the PS 300 Who Message Function. This subroutine then reads the reply from the PS 300 and sets the value of QBIN in COMMON BLOCK BINPAR. This subroutine also resets the host_message function and initializes the LEVEL 2 COMMON BLOCKS.

FORTRAN Calling Sequence:

```
PSCON (IN, IOUT, IODEV)
```

Where:

- IN (INTEGER*2) input logical unit number (Machine Dependent).
- IOUT (INTEGER*2) output logical unit number (Machine Dependent).
- IODEV (INTEGER*2) specifies the input/output device:

```
0) Low speed device.
1) High speed device.
3 .. n) other devices.
```

1.2.2 PSCOFF

This subroutine terminates the connection between the PS 300 and host I/O device.

FORTRAN Calling Sequence:

```
PSCOFF
```

1.2.3 PSSNDC

This subroutine loads a character buffer into a Queue Buffer for output to the PS 300. If the current Queue Buffer becomes full, it will be transmitted and another Queue Buffer will be loaded. These Queue Buffers will be routed through Ciroute Output #3 (routine byte = "0").

FORTRAN Calling Sequence:

```
PSSNDC (LEN, IOUTBF)
```

Where:

- **LEN** (INTEGER*2) Number of characters to output.
- **IOUTBF** (LOGICAL*1) Buffer of characters to output.

1.2.4 PSFINC

This subroutine sends out the last remaining Queue Buffer of characters (if there is one) to the PS 300. The Queue Buffer will be routed through Ciroute Output #3 (routine byte = "0"). The routine will clear the I/O output state flag.

FORTRAN Calling Sequence

```
PSFINC
```

1.2.5 PSSNDB

This subroutine loads a 16-bit array into a Queue Buffer for output to the PS 300. If the current Queue Buffer becomes full, it will be transmitted and another Queue Buffer will be loaded. These Queue Buffers will be routed through Ciroute Output #4 if the host device supports 8-bit data. Note: the most significant byte of the 16-bit word will be sent first.

FORTRAN Calling Sequence:

```
PSSNDB (LEN, IARRAY)
```

Where:

- **LEN** (INTEGER*2) Number of 16-bit numbers to output.
- **IARRAY** (INTEGER*2) Array of 16-bit numbers to output.

1.2.6 PSFINB

This subroutine sends out the last remaining Queue Buffer of 16-bit numbers (if there is one) to the PS 300. The Queue Buffer will be routed through Ciroute Output #4 if the host device supports 8-bit data. The routine will clear the I/O output state flag.

FORTRAN Calling Sequence:

```
PSFINB
```

1.2.7 PSFEXP

This subroutine finds the largest exponent of an array of REAL numbers.

FORTRAN Calling Sequence:

```
PSFEXP (N, ARRAY, MAXEXP)
```

Where:

- **N** (INTEGER*2) number of values to search.
- **ARRAY** (REAL) Array of values to be searched.
- **MAXEXP** (INTEGER*2) returned maximum exponent.

1.2.8 PSVNOR

This subroutine normalizes values from a REAL array into PS 300 vector-normalized format and stores the values into a 16-bit array.

FORTRAN Calling Sequence:

PSVNOR (N, ARRAY, IARRAY, MAXEXP)

Where:

- **N** (INTEGER*2) number of values to normalize.
- **ARRAY** (REAL) array of values to normalize.
- **IARRAY** (INTEGER*2) array to receive the normalized data.
- **MAXEXP** (INTEGER*2) the exponent used for normalization.

1.2.9 PSBNOR

This subroutine normalizes values from a REAL array into PS 300 block-normalized format and stores the values into a 16-bit array.

FORTRAN Calling Sequence:

PSBNOR (N, ARRAY, IARRAY, MAXEXP)

Where:

- N (INTEGER*2) number of values to normalize.
- ARRAY (REAL) array of values to normalize.
- IARRAY (INTEGER*2) array to receive the normalized data.
- MAXEXP (INTEGER*2) the exponent used for normalization.

1.2.10 PSRCVC

This routine receives a buffer of characters from the PS 300. The PS 300 message terminator character <CR> is stripped off.

FORTRAN Calling Sequence:

```
PSRCVC (INLEN, INBUF, NUMBYT)
```

Where:

- **INLEN** (INTEGER*2) maximum number of characters that can be received.
- **INBUF** (LOGICAL*1) array to receive the characters.
- **NUMBYT** (INTEGER*2) actual number of characters received.

1.2.11 PSTRHM

This subroutine sends a message to the PS 300 Ciroute output #18 (routine byte = "?") to trigger the Host Hold-Message Function.

FORTRAN Calling Sequence:

PSTRHM (MODE)

Where:

MODE (INTEGER*2) trigger mode:

0 = Flush all messages waiting to be sent to host; reset host_message.
1 = Send next message. If one is not waiting, send "No Message".
2 = Send next message. If one is not waiting, wait for one to come available.

1.2.12 PSPKWM

This subroutine sends a message to the PS 300 Ciroute output #19 (routine byte = "2") to poke the Who Message Function.

FORTRAN Calling Sequence:

```
PSPKHM
```

1.2.13 PSERR

This subroutine is called by other PSIO routines when an error is encountered.

FORTRAN Calling Sequence:

```
PSERR (N)
```

Where:

N (INTEGER*2) Error Number:

1) PSETUP has been called again before PSEXIT has been called.
2) A PSIO routine has been called before PSETUP has been called.
3) A PSIO routine which outputs to the PS 300 has been called before a Character Queue Buffer has been flushed.
4) A PSIO routine which outputs to the PS 300 has been called before a Binary Queue Buffer has been flushed.
5) A PSIO routine was called with a negative or zero buffer count.
6) PSVECS was called to output one type of vector list while in the middle of sending another type of vector list.
7) PSVECS was called with an invalid vector type.
8) A PSIO routine was called with an invalid intensity parameter.
9) PSVECS was called to output a Block Normalized vector list while in the middle of sending another Block Normalized vector list.

50..n) Machine Dependent Errors.

APPENDIX A. HOLD_MESSAGE

This appendix contains the function network diagram and functional description of HOLD_MESSAGE, an intrinsic function that supports the subroutines PSREAD and PSPOLL. This function is already part of the PS 300 System. All that is necessary is to connect to input 1 of HOST_MESSAGE.

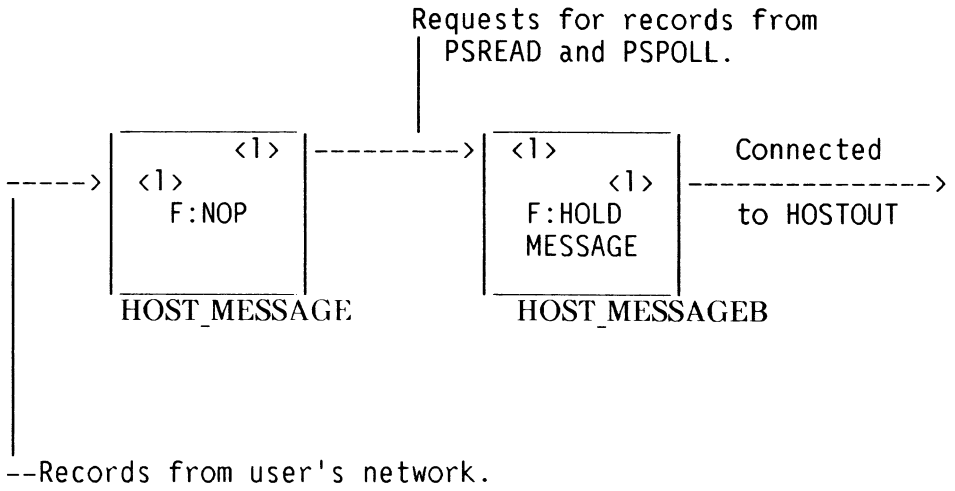

Figure A-1. Function Network Diagram

HOLD_MESSAGE

PROCEDURE Msghold

INPUTS

<1>: Qpackets of messages to be sent to the host. And Qintegers used to trigger the messages as follows:

Value of 0 - Clear any messages waiting in the private queue to be sent to the host.

Value of f1 If a message is waiting, send it. Otherwise send the "NO Message" Qpacket from input <3>.

Value of 2 If a message is waiting, send it. Otherwise set the Send Immediate Flag so that the next message will be sent immediately to the host.

<2>C: Message Terminator Qpacket that is added to the end of the input <1> message before it is sent to the host.

<3>C: "No Message" Qpacket.

Note: Inputs <2> and <3> default to carriage returns.

OUTPUTS

<1>: Qpacket to the Host Computer.

Private: Send Immediately Flag.

Queue of Qpackets waiting to be sent to the host.

APPENDIX B. RESERVED NAMES LIST

This appendix contains the reserved-word list. A word is reserved if it already names a subroutine, COMMON block, or function in the E&S-supplied Host I/O Subroutines. These words should not be used to name user functions.

SUBROUTINE AND FUNCTION NAMES

PSIO.FTN ,PSIO.FOR, VAXPSIO.FOR, and RSXPSIO.FTN

PSETUP
PSEXIT
PSPOLL
PSREAD
PSSEND
PSVECS
PSCHAR
PSFIXI
PSCON
PSCOFF
PSSNDC
PSFINC
PSSNDB
PSFINB
PSFEXP
PSERRC
PSBNOR
PSRCVC
PSTRHM
PSPKWM
PSERR

VAXPSSER.MAR and RSXPSSER.MAC

PSLSET
PSLOUT
PSLINQ
PSLIN
PSLFIN
PSLERR

VAXPSDMR.MAR and RSXDMR.MAC

PSHSET
PSHOUT
PSHIN
PSHFIN
PSHERR

COMMON AREA NAMES

PSIOFL
PSVNOR
PSBPAR
PSDEV
PSQUE
PSCIRT