# PS 300 DOCUMENT SET

# VOLUME 3a

## PROGRAMMER REFERENCE

The contents of this volume are not to be reproduced or copied in whole or in part without the prior written permission of Evans & Sutherland.

Many concepts in this volume are proprietary to Evans & Sutherland, and are protected as trade secrets or covered by U.S. and foreign patents or patents pending.

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.

PS1, PS2, MPS, and PS 300 are trademarks of the Evans & Sutherland Computer Corporation.

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

# VOLUME 3A

# PROGRAMMER REFERENCE

Volume 3A of the PS 300 User Documentation Set is a graphics programmer's reference to PS 300 commands and functions. Volume 3A and Volume 3B, which contains information on FORTRAN and Pascal Graphics Support Routines (GSRs), together constitute a complete source of reference materials for PS 300 programmers. Reference material of interest to the System Manager is contained in Volume 5.

This volume contains the following sections.

## PS 300 COMMAND SUMMARY

This document is a concise summary of information about the ASCII form of every command in the PS 300 Command Language set. The long form and acceptable short form of each command are given, together with information on parameters, default values, and other requirements. Where a command creates a node in a display tree, the type of node is indicated. If that node can be updated with values from an interactive device, the inputs to the node and acceptable data types are shown in a diagram. Examples of the use of commands are given whenever appropriate, and related information is included as notes. The summary is alphabetized for ease of use. Appendices list commands by classification, give the syntax of each command, and provide a cross-reference to the Graphics Support Routines in Volume 3B.

## PS 300 FUNCTION SUMMARY

This is a summary, in diagrams and text, of essential information about each function available to the user in the PS 300 intrinsic function and initial function instance set. Functions are represented as boxes with numbered input queues and outputs. Acceptable data types are indicated, as are default values and associated functions where appropriate. Notes explain any further features or peculiarities of functions, and examples of usage are often provided.

## PS 300 GRAPHICS FIRMWARE RELEASE NOTES

The Graphics Firmware Release Notes summarize the new features of the A1 release and list corrected problems, known problems, and miscellaneous notes and advice. They are specifically intended to point the user to revised documentation that discusses the firmware changes in detail. The Diagnostic Release Notes describe all new diagnostics and changes in the PS 300 diagnostic software.

## USER ERROR REPORTING AND INFORMATION MESSAGES

This document lists and explains all messages generated by the PS 300. Included are informational messages, warning messages, and non-fatal and fatal error messages.

# PS 300 COMMAND SUMMARY

Many concepts in this document are proprietary to Evans & Sutherland, and are protected as trade secrets or covered by U.S. and foreign patents or patents pending.

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

# PREFACE

This manual is a PS 300 Command Language reference for graphics programmers who are already familiar with the basic operation of the PS 300.

Commands are ordered alphabetically, with the command name in the upper right-hand corner of each page. The following information, where relevant, is given for each command:

- Name
- Category and sub-category
- Syntax
- Description
- Parameters
- Defaults
- Notes
- Display tree node created
- Inputs for updating node
- Notes on inputs
- Associated functions
- Examples

Appendix A shows how the commands are grouped into categories.

For a quick reference, Appendix B contains an alphabetical listing of just the command syntax.

Appendix C contains a cross-reference between the ASCII form of the commands and the Graphics Support Routines.

Since some commands require the ASCII decimal equivalent of characters in their parameters, an ASCII chart with decimal values is included after the appendices.

# PS 300 COMMAND SUMMARY

## CONTENTS

## PS 300 COMMAND SUMMARY

# PS 300 COMMAND SUMMARY

## Command Syntax

A command's syntax is shown at the top of the page. In the syntax, UPPERCASE letters are required and lowercase letters are optional. Command parameters are shown in the syntax in **boldface.** Parameters are optional if enclosed in [square brackets], and required otherwise.

There are two main types of PS 300 commands - data structuring commands and immediate action commands.

## Data Structuring Commands

The data structuring commands are the only commands that can be named either directly or indirectly (by being included in a BEGIN_STRUCTURE ... END_STRUCTURE). These commands are named because they create nodes in a display structure (display tree) in mass memory. These nodes have to be accessed, and the name given to the command which creates a node is the address of that node in memory.

## Immediate Action Commands

Immediate action commands cannot be named. These commands perform immediate operations and do not create nodes in mass memory. In other words, there is nothing to associate an address (name) with.

## FORMAT

```
name := ALLOCATE PLOTTER device_number;
```

## DESCRIPTION

Allows you to specify which of up to four plotters to allocate in order to obtain hardcopies of the currently displayed PS 300 screen image. It also supresses automatic form feeds between plots.

## PARAMETERS

**device_number** - An integer between 0 and 3 which indicates the device number of the plotter you want to allocate.

## NOTE

The main use of this command is to supress automatic form feeds between plots.

## DISPLAY TREE NODE CREATED

ALLOCATE PLOTTER operation node.

## INPUTS FOR UPDATING NODE

None.

## FORMAT

```
name := operation_command [APPLied to name1];
name := operation_command [THEN name1];
```

## DESCRIPTION

Associates a command to the structure which is to be affected by the command.

## PARAMETERS

**operation_command** – A command that creates an operation node in a display tree.

**name1** – Structure that will be affected by the command.

## NOTE

APPLied to and THEN are synonyms. The terms are completely interchangeable.

## DISPLAY TREE NODE CREATED

The command node with a pointer to the structure **name1**.

## EXAMPLE

```
A:= ROTate in X 45 THEN B;

B:= VECtor_list n=5 1,1 -1,1 -1,-1 1,-1 1,1;
```

## FORMAT

name := ATTRIBUTES **attributes** [AND **attributes**];

## DESCRIPTION

Specifies the various characteristics of polygons used in the creation of shaded renderings. This command is only used with the PS 340. For a detailed explanation of defining and interacting with shaded images, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2.

## PARAMETERS

**attributes** - The attributes of a polygon are defined as follows.

[COLOR **h**[,**s**[,**i**]]] [DIFFUSE **d**] [SPECULAR **s**]

where

**h** - is a real number specifying the hue in degrees around the color wheel. Pure blue is 0 and 360, pure red is 120, and pure green is 240.

**s** - is a real number specifying saturation. No saturation (gray) is 0 and full saturation (full toned colors) is 1.

**i** - is a real number specifying intensity. No intensity (black) is 0, full intensity (white) is 1.

**d** - is a real number from 0 to 1 specifying the proportion of color contributed by diffuse reflection versus that contributed by specular reflection. Increasing **d** makes the surface more matte. Decreasing **d** makes it more shiny.

**s** - is an integer from 0 to 10 which adjusts the concentration of specular highlights. The more metallic an object is, the more concentrated the specular highlights.

(continued)

## DEFAULTS

If no color is specified, the default is white (s = 0, i = 1). If saturation and intensity are not specified, they default to 1. If only hue and saturation are specified, intensity defaults to 1. If no diffuse attribute is given, **d** defaults to .75. If no specular attribute is given, s defaults to 4.

## NOTES

1. Polygon attribute nodes are created in Mass Memory but are not part of a display tree. The attributes specified in an ATTRIBUTES command are assigned to polygons which include a WITH ATTRIBUTES clause. The attributes specified in a WITH ATTRIBUTES clause of a polygon command apply to all subsequent polygons until superseded by another WITH ATTRIBUTES clause. If no WITH ATTRIBUTES option is given for a polygon node, default attributes are assumed. The default attributes are 0,0,1 for color, 0.75 for diffuse, and 4 for specular.

2. The various attributes may be changed from a function network via inputs to an attribute node, but the changes have no effect until a new rendering is created.

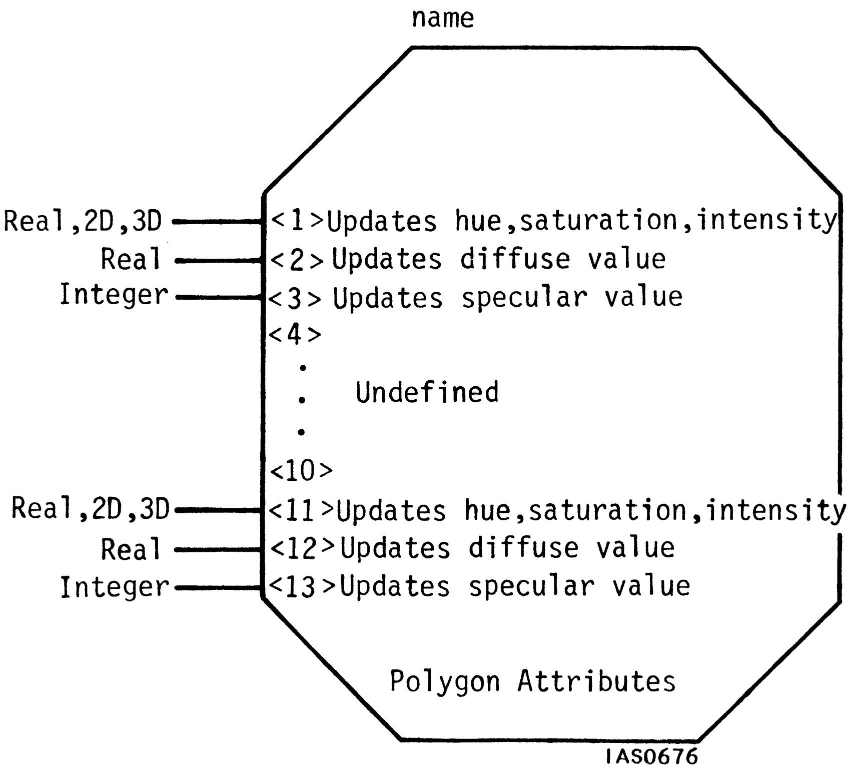
3. A second set of attributes may be given after the word AND in the ATTRIBUTES command. These attributes apply to the obverse side of the polygon(s) concerned. In other words, the two sides of an object may have different attributes. The attributes defined in the first **attributes** pertain to front-facing polygons. Those in the AND **attributes** clause pertain to backfacing polygons.

## NODE CREATED

Polygon attribute definition node. This node resides in Mass Memory, but is not included in a display tree.

(continued)

## INPUTS FOR UPDATING NODE


name
Real,2D,3D
Real
Integer
<1>Updates hue,saturation,intensity
<2>Updates diffuse value
<3> Updates specular value
<4>
Undefined
<10>
Real,2D,3D
Real
Integer
<11>Updates hue,saturation,intensity
<12>Updates diffuse value
<13>Updates specular value
Polygon Attributes
IAS0676

## NOTES ON INPUTS

1. Inputs <1> and <11> accept a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation and intensity.

2. Values sent to inputs <1>, <2>, and <3> specify the COLOR for the front of the polygon(s) or for both sides if no obverse attributes are given.

3. If anything other than a 3D vector is sent to input <1> or <11>, default values for the other variables are assumed.

## FORMAT

```
name := BEGIN_Font
            [C[0]: N=n {itemized 2D vectors};]
               .
               .
               .
            [C[i]: N=n {itemized 2D vectors};]
               .
               .
               .
            [C[127]: N=n {itemized 2D vectors};]
        END_Font;
```

## DESCRIPTION

Defines alternative character fonts, using itemized 2D vector lists to describe each character. Up to 128 PS 300 character codes may be defined for each font.

## PARAMETERS

n - Number of vectors in 2D vector list.

i - Decimal ASCII code to be defined. The square brackets around the ASCII number from 0 to 127 are required.

{itemized 2D vectors} - Vectors making up the ASCII character being defined (P x1, y1, L x2, y2, etc).

## NOTES

1. Not all ASCII codes need to be defined for a font. Nothing is output for an undefined character.

2. There is no restriction on the range of values for the 2D vector making up a character, but for correct spacing and orientation to adjacent characters, the range in x and y should be kept between 0 and 1.

## NODE CREATED

Alternate character font definition node. This node resides in Mass Memory but is not part of a display tree. To specify an alternate font, the character FONT command is used. This creates a character FONT node in a display tree which points to the appropriate alternate font definition.

## INPUTS FOR UPDATING NODE

None.

## EXAMPLE

```
A := BEGIN_Font
        C[65]: N=5 P 0,0 L .9,0 L .9,.9 L 0,.9 L 0,0;
     END_Font;

B := BEGIN_Structure
        character FONT A;
        CHARacters 'ABA';
     END_Structure;

DISPlay B;
```

{Two squares - the new A - will appear right next to each other with the lower left corner of the first at the origin. The letter B is not defined in character FONT A, so nothing is DISPlayed for B. Note that this exam;le creates a special symbol (a square) rather then defining an alternate character font.}

## FORMAT

```
         name := BEGIN_Structure
[name1:=] nameable_command;
                    .
                    .
                    .
[namen:=] nameable_command;
          END_Structure;
```

## DESCRIPTION

Groups a set of viewing and/or modeling commands so that each element does not need to be explicitly named and APPLied to the next structure in line. This does not, however, prevent naming nested commands directly or explicitly applying a command to another structure via APPLied to.

## PARAMETERS

**name1..namen** - Optional names for individual commands inside the BEGIN_S...END_S, allowing reference to these specific commands from elsewhere (see Note 3). The PS 300 prefixes these names with the name of the outer structure and a period (.), ad infinitum. So, for example, the command defined as **name1** in the structure is referenced as **name.name1.**

**nameable_command** - Nameable commands are those that can be prefixed with "name :=", with the following exceptions:

- COMmand STATus can also be used.
- Intrinsic Functions cannot be instanced.
- name := nil; cannot be used.

(continued)

## NOTES

1. Essentially, any data structuring command except a function instancing command can be used.

2. A non-data command inside a BEGIN_S...END_S is applied to every node that follows in the structure unless it is explicitly APPLied to another structure, in which case it only affects the APPLied to structure (see examples).

3. If a command inside the structure is to be modified later by a function network or from the host, it must be named so that it can be referenced. Its referencing name is the name with all prefixes (e.g. name.name1).

## DISPLAY TREE NODE CREATED

The various nodes created by the "nameable commands" linked together as specified. The top node of this structure is name and is an instance node.

## INPUTS FOR UPDATING NODE

The nodes that may be updated are created by those nameable commands that are explicitly named (see note 3). For inputs, refer to the individual command descriptions.

(continued)

EXAMPLES

```
A:= BEGIN_Structure
     TRANslate by 2,3;
     BEGIN_Structure
               ROTate 30;
               SCALE .5 THEN B;
     END_Structure;
     VECtor_list ... ;
Rot:= ROTate in X 45 THEN C;
     ROTate in Y 90;
     character FONT D THEN E;
Char:= CHARacters 'ABC';
Dat:=  VECtor_list ... ;
     END_Structure;
```

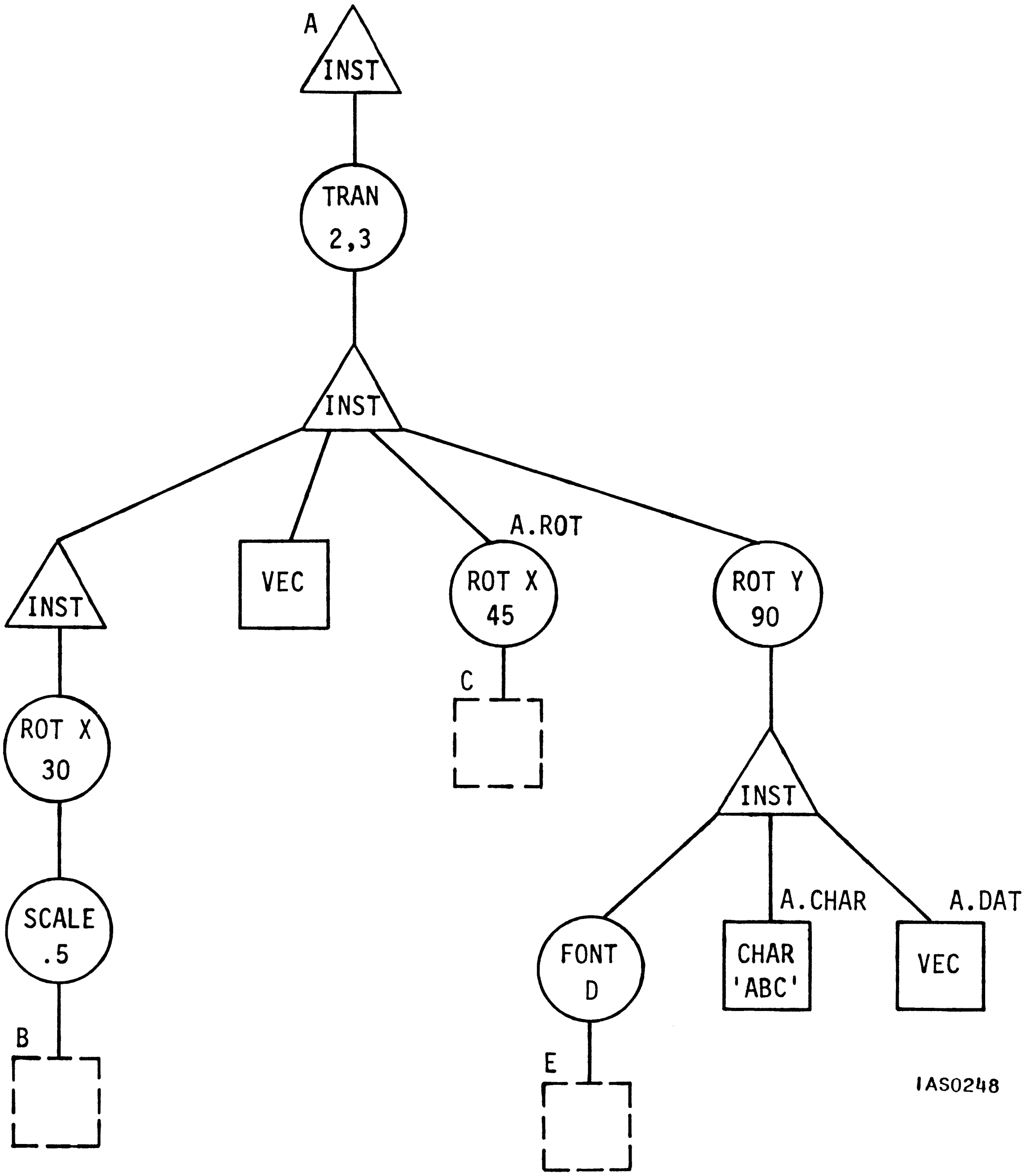
{To modify the X angle of rotation, a 3x3 matrix would be sent to <1>A.rot. You could not modify the Y rotation angle since it is not explicitly named.}

{An equivalent display tree could be created without using BEGIN_Structure ... END_Structure, for example:}

```
A:= INSTance of F;
F:= TRANslate by 2,3 THEN G;
G:= INSTance of H,I,A.ROT,J
H:= INSTance of K;
I:= VECtor_list ...;
A.ROT:= ROTate in X 45 THEN C:
J:= ROTate in X 90 THEN L;
K:= ROTate in Y 30 THEN M;
L:= INSTance of N,A.CHAR,A.DAT;
M:= SCALE .5 THEN B;
N:= character FONT D THEN E;
A.CHAR:= CHARacters 'ABC';
A.DAT:= VECtor_list ... ;
```

(continued)

EXAMPLES



## FORMAT

```
BEGIN
command;
command;
  .
  .
  .
command;
END;
```

## DESCRIPTION

Defines a "batch" of commands which take effect in a single screen update, so that they appear to be executed simultaneously.

## PARAMETERS

command - Any PS 300 command.

## NOTE

Although any commands may be used inside a BEGIN...END structure, only commands that create, display, or delete objects will happen "simultaneously".

## EXAMPLE

```
BEGIN
DISPlay A;
A:= VECtor_list n=5 1,1 -1,1 -1,-1 1,-1 1,1;
DISPlay B;
B:= VECtor_list n=4 0,0 1,0 1,1 0,0;
END;
```

{A and B will be displayed simultaneously.}

## FORMAT

```
name := BSpline ORDER= k
        [OPEN/CLOSED] [NONPERIodic/PERIodic] [N= n]
        [VERTICES =] x1,y1,z1
                     x2,y2,z2
                      .  .  .
                      .  .  .
                      .  .  .
                     xn,yn,zn
        [KNOTS] = t1,t2,...,tj
        CHORDS = q;
```

## DESCRIPTION

Evaluates a B-spline curve, allowing the parametric description of the curve form without the need to specify or transfer the coordinates of each constituent vector.

The B-spline curve C is defined as:

$$C(t) = \sum_{i=1}^{n} p_i N_{i,k}(t)$$

where

$p_i$ - ith vertex of the B-spline's defining polygon

and

$N_{i,k}$ - ith B-spline blending function of order k.

The parameter t of the curve and blending functions is defined over a sequence of knot intervals $t_1, t_2, \ldots, t_{n+k}$. Different knot sequences define different types of B-splines. Two common knot sequences are uniform non-periodic and uniform periodic. A uniform non-periodic B-spline is defined by the knot sequence:

$$t_j = \begin{cases} 0 & (\text{for } j < k) \\ j-k & (\text{for } k < j < n) \\ n-k+1 & (\text{for } n < j < n+k) \end{cases}$$

A uniform periodic B-spline is defined by the knot sequence:

$t_j = j$ (for $j < n+k$)

(continued)

## DESCRIPTION (continued)

The blending functions can be defined recursively as

$Ni,1(t) = 1$ ( if $ti < t < ti+1$), 0 otherwise

$$Ni,k(t) = \frac{(t-ti)Ni,k-1(t)}{ti+k-1-ti} + \frac{(ti+k-t)Ni+1,k-1(t)}{ti+k-ti+1}$$

The curve is evaluated at the points:

$$t = \frac{(1-i)ti + itj-k+1}{q}$$

for $i=0,1,2,\ldots,q$.

## PARAMETERS

k - The order of the curve ($0 < k$).

n - The number of vertices (used to anticipate storage requirements).

x1,y1,z1...xn,yn,zn - The vertices of the defining polygon of the curve. The z component is optional.

t1,t1,...,tj - User specified knot sequence. Because closed B-splines are evaluated as open B-splines with duplicate vertices, the number of knots required is:

| | |
|---|---|
| n+k | for open B-splines |
| n+k+1 | for closed non-periodic B-splines |
| n+2k-1 | for closed periodic B-splines |

The knots must also be non-decreasing.

q - The number of vectors to be created ($0 < q \leq 32767$).

(continued)

NOTES

1. OPEN or CLOSED is an option which describes the B-spline defining polygon. The default is OPEN. (Note that CLOSED merely describes the polygon, eliminating repetition of the last vertex.)

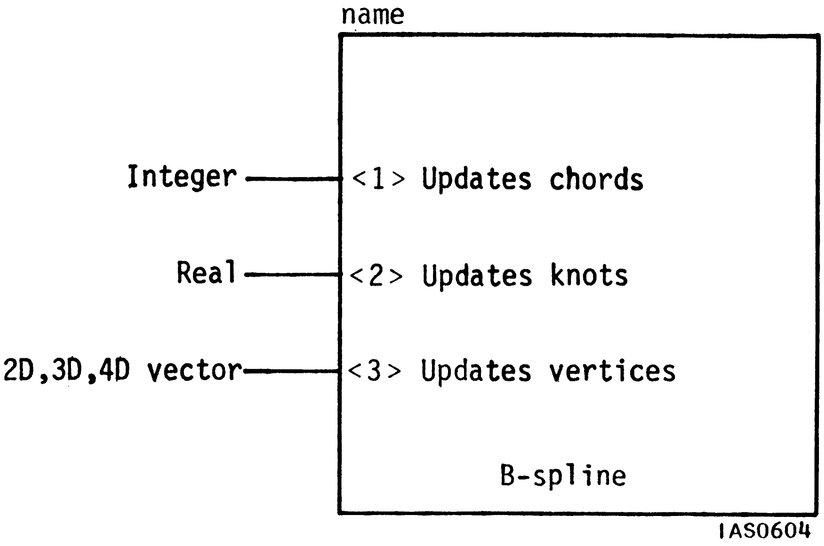
2. If no knot sequence is given, NONPERIODIC or PERIODIC is an option which specifies that the non-periodic or periodic knot sequence be used as the knot sequence. NONPERIODIC is the default for open B-splines; PERIODIC is the default for closed B-splines.

3. At least k vertices must be given, or the order k will be reduced accordingly.

DISPLAY TREE NODE CREATED

B-spline vector list data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUT

1. The z value of a vector defaults to 0 when a 2D vector is sent to a 3D B-spline.

2. W and z values should be ignored when a 3D or 4D vector is sent to a 2D Bspline.

## FORMAT

```
name := character FONT font_name [APPLied to name1];
```

## DESCRIPTION

Establishes a user-defined alternate character font as the working font. This font must have been previously defined with . the BEGIN_Font ... END_Font command. If the font is not defined, the current font is still used.

## PARAMETERS

font_name - Name of the desired font.

name1 - Structure to use the character font.

## DISPLAY TREE NODE CREATED

Character font operation node. This node points to the definition of the alternate font that is to be used.

## INPUTS FOR UPDATING NODE

None.

## EXAMPLE

```
New_Font := BEGIN_Font

                {character definitions}
              END_Font

A := BEGIN_Structure
          CHARacters 'HERE'; {this uses standard font}
          character FONT New_font;
          CHARacters 0,-2 'HERE'; {this uses the font New_Font}
     END_Structure;

DISPlay A;
```

## FORMAT

name := CHARacter ROTate **angle** [APPLied to **name1**];

## DESCRIPTION

Rotates characters. Creates a 2x2 rotation matrix to be applied to the specified characters (in **name1**).

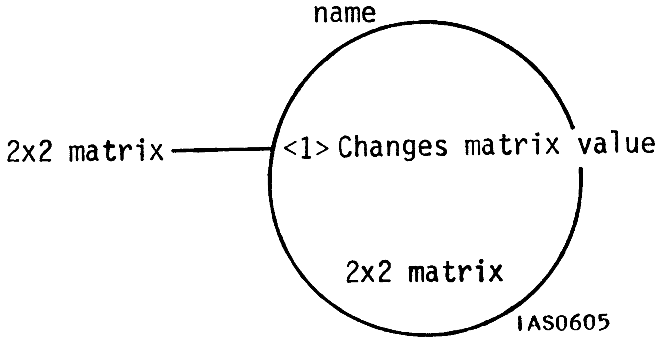
## PARAMETERS

**angle** - Z-rotation angle in degrees (unless other units are specified). When you are looking along the positive direction of the Z axis, positive **angle** values produce counterclockwise rotations.

## DISPLAY TREE NODE CREATED

2x2 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 2x2 matrix is legal.

(continued)

## ASSOCIATED FUNCTIONS

F:MATRIX2, F:CROTATE, F:CSCALE

## EXAMPLE

```
A:= CHARacter ROTate 90 THEN B;

B:= CHARacters 'Vertical';
```

{If A were DISPlayed, the text Vertical would start at the origin and read up the Y axis.}

## FORMAT

```
name := CHARacter SCAle s [APPLied to name1];
name := CHARacter SCAle sx,sy [APPLied to name1];
```

## DESCRIPTION

Creates a uniform (s) or non-uniform (sx,sy) .2x2 scale matrix to scale the specified characters.

## PARAMETERS

s - Scaling factor for both axes.

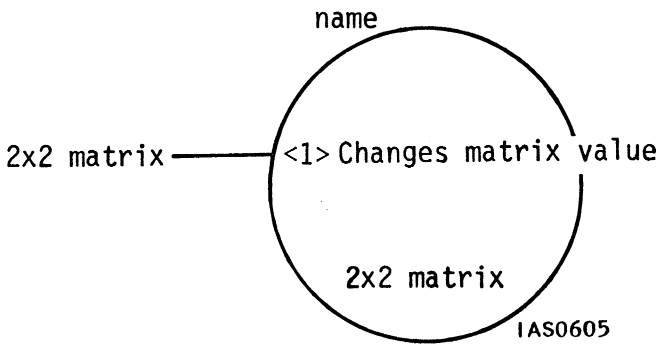
sx,sy - Separate axial scaling factors.

name1 - Structure whose characters are to be scaled (vector lists in the structure are not affected).

## DISPLAY TREE NODE CREATED

2x2 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 2x2 matrix is legal.

(continued)

ASSOCIATED FUNCTIONS

F:MATRIX2, F:CROTATE, F:CSCALE

EXAMPLE

```
A:= CHARacter SCAle .5 THEN B;

B:= CHARacters 'Half scale';
```

## FORMAT

```
name := CHARacters [x,y[,z]][STEP dx,dy] 'string';
```

## DESCRIPTION

Displays character strings and (optionally) specifies their location and placement.

## PARAMETERS

x,y,z - Location in the data space of the beginning of the character string (i.e., the lower left corner of a box enclosing the first character).

dx,dy - Spacing between the characters, in character size units. The width of the character is one dx unit; the height is one dy unit.

string - Text string to be displayed (up to 240 characters).

## DEFAULT

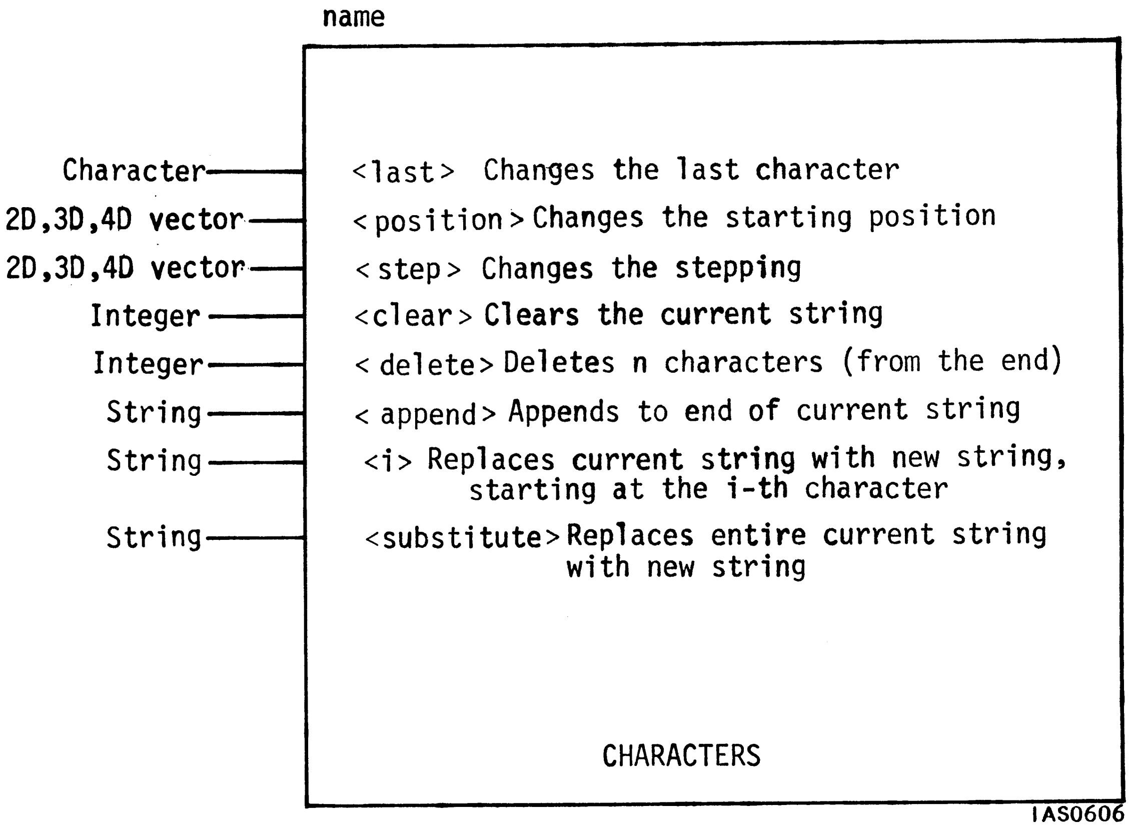
If string is the only parameter specified, the character string will start at 0,0,0 and dx,dy will be 1,0 (i.e., regular horizontal spacing).

## DISPLAY TREE NODE CREATED

Characters data node.

(continued)

INPUTS FOR UPDATING NODE


name
Character
2D,3D,4D vector
2D,3D,4D vector
Integer
Integer
String
String
String
<last> Changes the last character
<position> Changes the starting position
<step> Changes the stepping
<clear> Clears the current string
<delete> Deletes n characters (from the end)
<append> Appends to end of current string
<i> Replaces current string with new string, starting at the i-th character
<substitute> Replaces entire current string with new string
CHARACTERS
IAS0606

EXAMPLES

```
CHARacters 'HERE';

CHARacters 3,-3 STEP .5,1 'HERE';

CHARacters STEP -1,0 'HERE';
```

## FORMAT

```
COMmand STATus;
```

## DESCRIPTION

Used with BEGIN...END and BEGIN_STRUCTURE...END_STRUCTURE commands to report the current level to which these structures are nested.

## PARAMETERS

None.

## NOTES

1. If a syntactically correct command produces a parser syntax error, there may be unENDed BEGINs or BEGIN_STRUCTUREs causing the PS 300 to expect one or more ENDs or END_STRUCTUREs. By sending COMMAND STATUS, you can see if this is the case.

2. The !RESET command can be used to get out of unended BEGIN's or BEGIN_STRUCTURE's when a problem occurs, (see !RESET).

## FORMAT

CONNect name1<i>:<j>name2;

## DESCRIPTION

Connects function instance name1's output <i> to input <j> of function instance or display tree node name2.

## PARAMETERS

name1 – Function instance to be connected from.

<i> – Output number of function instance name1 to be connected. Refer to the PS 300 Function Summary for specific functions and acceptable values.

name2 – Function instance or display tree node to be connected to.

<j> – Input number or input name (in the case of some display tree nodes) of name2 to be connected. Refer to the PS 300 Function Summary for specific functions and acceptable values.

## FORMAT

name := COPY **name1** [START=] **i** [,] [COUNT=] **n**;

## DESCRIPTION

Creates a VECtor_list node containing a group of consecutive vectors copied from another vector list (**name1**) or a LABELS node containing a group of consecutive labels from an existing block (**name1**).

## PARAMETERS

**name** – Name of new VECtor_list or LABELS node.

**name1** – Name of the node being copied from.

**i** – First vector or index of first label in **name1** to be copied.

**n** – Last vector or count of labels in **name1** to be copied.

## NOTE

The keywords START= and COUNT= are optional, but if one is used, both must be used.

## DISPLAY TREE NODE CREATED

VECtor_list or LABELS data node.

## INPUTS FOR UPDATING NODE

(See VECtor_list or LABELS command).

(continued)

## EXAMPLES

```
A := VECtor_list n=5 .5,.5 -.5,.5 -.5,-.5 .5,-.5 .5,.5;

B := COPY A 1 3;

     {This would be the same as saying:
      B := VECtor_list n=3 .5,.5 -.5,.5 -.5,-.5;}

C := COPY A START=2 , COUNT=2;

     {This would be the same as saying:
      C := VECtor_list n=2 -.5,.5 -.5,-.5;}
```

## FORMAT

```
name := DEALLOCATE PLOTTER device_number;
```

## DESCRIPTION

Allows you to specify which of up to four plotters to deallocate after hardcopies of the currently displayed PS 300 screen image have been plotted. Enables automatic form feeds between plots.

## PARAMETERS

device_number - An integer between 0 and 3 which indicates the device number of the plotter you want to deallocate.

## DISPLAY TREE NODE CREATED

DEALLOCATE PLOTTER operation node.

## INPUTS FOR UPDATING NODE

None.

## FORMAT

name := DECrement LEVel_of_detail[APPLied to name1];

## DESCRIPTION

Decrements the current level of detail by 1 when **name** is being traversed.

## PARAMETERS

**name1** – Structure to be affected by the decreased level of detail.

## NOTE

There is really only one global level of detail; this command only changes the value of the level of detail while the named node and nodes below it in a display tree are being traversed.

## DISPLAY TREE NODE CREATED

DECREMENT LEVEL_OF_DETAIL operation node.

## INPUTS FOR UPDATING NODE

None.

(continued)

EXAMPLE

```
A:= SET LEVel_of_detail TO 5 THEN B;

B:= BEGIN_Structure
    IF LEVel_of_detail = 4 THEN C;
    IF LEVel_of_detail = 5 THEN D;
    DECrement LEVel_of_detail;
    IF LEVel_of_detail = 4 THEN E;
    IF LEVel_of_detail = 5 THEN F;
    END_Structure;

{If A were DISPlayed, structures D and E would also be displayed.}
```

## FORMAT

```
DELete name[,name1 ... namen];
DELete any_string*;
```

## DESCRIPTION

Sets **name** to nil, then FORGETs **name**. The wild card delete will set to nil any name beginning with the string that is entered.

## PARAMETERS

**name** - Any previously-defined name.

**any_string** - A character string which is part of any **name**.

## NOTES

1. After a DELete **name** command is issued, all Function Instances and structures referring to **name** will no longer include the data formerly associated with **name**.

2. After a DELete **name** command is issued, further definitions of or references to **name** will not change structures which referred to **name** before the DELete.

3. Compare with FORGET, which eliminates **name** while preserving objects which it formerly referred to.

4. If the wild card delete is used on an object being displayed, the object must be removed from display before entering the wild card delete command. Failure to do this will results in a small amount of memory being used for each object still displayed.

5. If a name is created from the host, it must be deleted via the host line. Similarly, if a name is created locally using the keyboard, the DELete command must be entered locally.

## FORMAT

DISCONNect name1[<i>]:option;

## DESCRIPTION

Disconnects one or all of Function Instance name1's outputs from one or all inputs that it has previously been connected to.

## PARAMETERS

name1 – Function Instance to disconnect output(s) from.

<i> – The output number of name1 to disconnect. If this is not specified, all of name1's outputs are implied and the option parameter must be ALL (this would disconnect all of name1's outputs from everything they had previously been connected to).

option – Either the keyword ALL or <j>name2, where:

ALL – Disconnect the specified output of name1 (or all outputs of name1) from all Function Instances or display tree nodes that it was previously connected to.

<j> – Input number or input name of name2 to be disconnected from name1.

name2 – Function instance or named node previously connected to name1.

## FORMAT

DISPlay **name**;

## DESCRIPTION

Displays a structure. Adds **name** to the Display Processor's display list.

## PARAMETERS

**name** - Any structure name.

## FORMAT

ERASE PATTERN FROM **name**;

## DESCRIPTION

An immediate action command which erases a pattern from a vector list (**name**).

## PARAMETERS

**name** – The vector list containing the pattern you want to erase.

## DISPLAY TREE NODE CREATED

None.

## FORMAT

```
name := EYE BACK z [option1][option2] from SCREEN area w WIDE
        [FRONT boundary = zmin BACK boundary = zmax]
        [APPLied to name1];
```

## DESCRIPTION

Specifies a viewing pyramid with the eye at the apex and the frustum of the pyramid (bounded by zmin and zmax) enclosing a portion of the data space to be displayed in perspective projection. Unlike the Field_Of_View command, the EYE command can create a skew (non-right) viewing pyramid (compare Field_Of_View and WINDOW).

## PARAMETERS

z - The perpendicular distance of the eye from the plane of the viewport.

option1 - RIGHT x or LEFT x, where x is the distance of the eye right or left of the viewport center, respectively, in relative room coordinates.

option2 - UP y or DOWN y, where y is the distance of the eye up or down from the viewport center, respectively, in relative room coordinates.

w - Width of the viewport in relative room coordinates.

zmin,zmax - Front and back boundaries of the frustum of the viewing pyramid. (See note 3 of the LOOK command for properly specifying zmin and zmax.)

name1 - Structure to which the EYE viewing area is applied.

## DEFAULT

None. If no EYE is specified, the default WINDOW is assumed (parallel projection X = -1:1 Y = -1:1 FRONT = $10^{-15}$ BACK = $10^{15}$). Refer to the WINDOW command.

NOTES

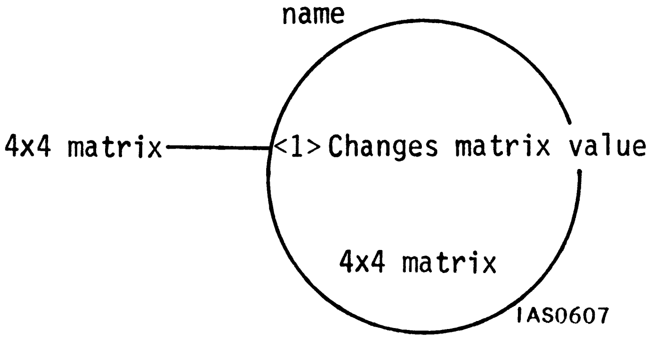
1. Notice that EYE always creates square side boundaries because the viewport width (w) is also taken to be the height; the aspect ratio is always 1.

2. If x and y are not specified (i.e. 0), then a right rectangle viewing pyramid is created (compare FOV).

DISPLAY TREE NODE CREATED

4x4 matrix operation node.

INPUTS FOR UPDATING NODE



ASSOCIATED FUNCTIONS

F:FOV, F:WINDOW, F:MATRIX4

EXAMPLE

```
A:= BEGIN_Structure
     EYE BACK 24 LEFT 1.5 FROM SCREEN area 10 WIDE
            FRONT boundary = 12
            BACK boundary = 14;
     LOOK AT 0,0,0 FROM 5,6.63,-10;
     INSTance of SPHERE;
     END_Structure;
```

{If SPHERE is defined with a radius of 1 about the origin, A would be a view of the SPHERE from 5,6.63,-10 fully depth-cued. Note that the FROM to AT distance in the LOOK AT command is 13.}

## FORMAT

```
name := Field_Of_View angle
           [FRONT boundary = zmin BACK boundary = zmax]
           [APPLied to name1];
```

## DESCRIPTION

Specifies a right rectangular viewing pyramid with the eye at the apex and the frustum of the pyramid (bounded by **zmin** and **zmax**) enclosing a portion of the data space to be displayed in perspective projection (compare EYE and WINDOW).

## PARAMETERS

**angle** - Angle of view from the eye (i.e., the FROM point established in the LOOK command) in x and y. (See note 1 below.)

**zmin,zmax** - Front and back boundaries of the frustum of the viewing pyramid. (See note 3 of the LOOK command for properly specifying **zmin** and **zmax**.)

**name1** - Structure to which the **FOV** is applied.

## DEFAULT

None. If no Field_Of_View is specified, the default WINDOW is assumed instead (parallel projection X = -1:1 Y = -1:1 **FRONT** = $10^{-15}$ **BACK** = $10^{15}$). Refer to the EYE command.

## NOTES

1. Notice that FOV always creates square side boundaries because **angle** defines both the x and the y angles; the aspect ratio is always 1.

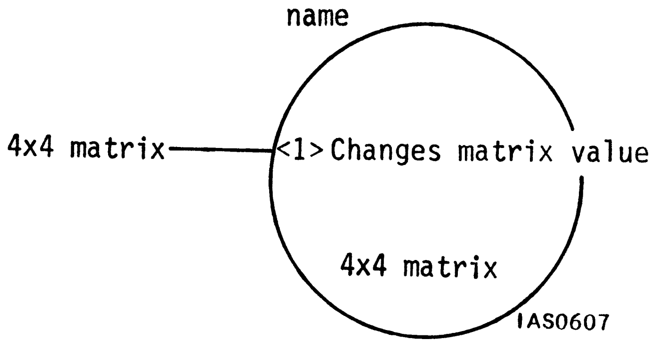
2. See also notes for the WINDOW command.

## DISPLAY TREE NODE CREATED

4x4 matrix operation node.

(continued)

## INPUTS FOR UPDATING NODE



## ASSOCIATED FUNCTIONS

F:FOV, F:WINDOW, F:MATRIX4

## EXAMPLE

```
BEGIN_Structure
     Field_Of_View 30
          FRONT boundary 12
          BACK boundary 14;
          LOOK AT 0,0,0 FROM 5,6.63,-10;
          INSTance of SPHERE;
END_Structure;
```

{If SPHERE is defined with a radius of 1 about the origin, A would be a view of the SPHERE from 5,6.63,-10 fully depth-cued. Note that the FROM to AT distance in the LOOK command is 13.}

## FORMAT

FOLLOW **name** WITH **option**;

## DESCRIPTION

Follows a named operation node (**name**) with another operation node.

## PARAMETERS

**name** - A named transformation, attribute, or conditional reference node to be followed with one of the options.

**option** -

1. A node created by a transformation command (SCALE by, ROTate, etc).
2. A node created by an attribute setting command (SET LEVel_of_detail, etc.).
3. A node created by a conditional referencing command (IF LEVel_of_detail, etc).

## NOTE

The structure **name** does not change association, unlike a named structure in a PREFIX WITH command.

## DISPLAY TREE NODE CREATED

An operation node corresponding to the option phrase of the command. This node points to whatever node **name** pointed to previously. The node is also pointed to by **name**.

(continued)

EXAMPLE

```
SHAPE :=  BEGIN_Structure
          tran := TRANslate by 20,20;
          rotate := ROTate in X 90;
          triangle := VECtor_list n=4 0,0 0,3 3,0 0,0;
          END_Structure;
FOLLOW SHAPE.ROT WITH SCALE by 2;
```

{This will alter the structure SHAPE so that SHAPE.triangle is first scaled, then rotated, then translated.}

## FORMAT

FORget **name**;

## DESCRIPTION

If the structure **name** is being displayed, it is removed from the display. **name** is also removed from the name dictionary.

## PARAMETERS

**name** - Any previously-defined structure name.

## NOTES

1. After a FORget name command is issued for a structure, all Function Instances and structures referring to **name** will continue to refer to the data formerly associated with **name**, even though **name** is no longer linked with the data.

2. After a FORget **name** command is issued for a structure, further definitions of, or references to, **name** will not change structures which referred to **name** before the FORget command.

3. Compare with DELete, which affects not only **name** but the content of **name** also.

## FORMAT

```
FORget (unit_name);
```

## DESCRIPTION

Removes a unit definition from memory.

## PARAMETERS

**unit_name** - Any previously-assigned unit name.

## NOTE

Note that FORget requires unit names to be enclosed in parentheses (unlike structure names).

## FORMAT

```
name := F:function_name;
```

## DESCRIPTION

Creates an instance of PS 300 Intrinsic Function.

## PARAMETERS

**name** – Any combination of alphanumeric characters up to 240. Must begin with an alpha character and can include $ or _.

**function_name** – Any PS 300 Intrinsic Function name.

## EXAMPLE

```
Add1 := F:add;
Add2 := F:add;
```

{This creates two different instances of the same Intrinsic Function F:add.}

## FORMAT

name := IF conditional_BIT n is state [THEN name1];

## DESCRIPTION

Refers to a structure if an attribute bit has a specified setting (ON or OFF). (See SET conditional_BIT command).

## PARAMETERS

n - Integer from 0 to 14 indicating which bit to test.

state - The setting to be tested (ON or OFF).

name1 - Structure to be conditionally referenced.

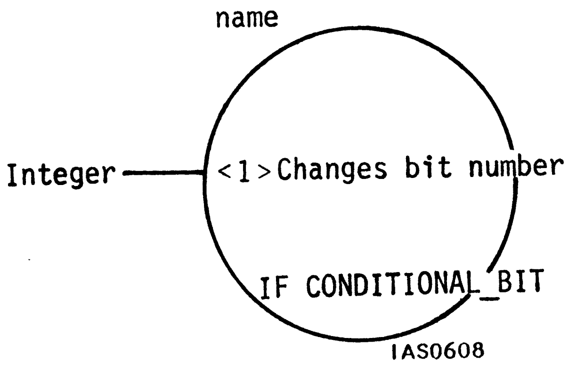
## DEFAULT

If bit n was not manipulated higher in the display tree, it will default to OFF.

## DISPLAY TREE NODE CREATED

IF CONDITIONAL_BIT operation node (conditional connection between two structures).

## INPUTS FOR UPDATING NODE



(continued)

## NOTES ON INPUTS

Input <1> accepts an integer (between 0-14) to change the bit number to the integer value.

## EXAMPLE

```
A:= SET conditional_BIT 3 ON THEN B;

B:= IF conditional_BIT 3 is ON THEN C;

C:= VECtor_list ... ;
```

{Initially when A is DISPlayed, C would also be displayed, indirectly. If a function network were connected to A to change conditional bit 3 to OFF, then the test in B would fail and C would not be displayed.}

## FORMAT

name := IF LEVel_of_detail **relationship n** [THEN **name1**];

## DESCRIPTION

Refers to a structure if the level of detail attribute has a specified relationship to a given number. Tests the relation between the current level of detail and the number **n** (see SET LEVel_of_detail command).

## PARAMETERS

**relationship** – The relationship to be tested (<, <=, =, <>, >=, >).

**n** – Integer from 0 to 32767 indicating the number to compare the current level of detail to.

**name1** – Structure to be conditionally referenced.

## DEFAULT

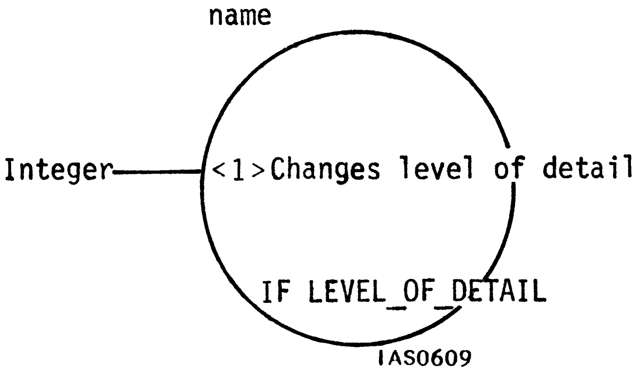
If the level_of_detail is not manipulated higher in the structure by a SET LEVel_of_detail node, it will default to 0.

## DISPLAY TREE NODE CREATED

IF LEVEL_OF_DETAIL operation node (conditional connection between two structures).

(continued)

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

Input <1> accepts an integer (from 0 to 32767) to change the level of detail to the integer value.

## EXAMPLE

```
A:= SET LEVel_of_detail to 3 THEN B;

B:= IF LEVel_of_detail = 3 THEN C;

C:= VECtor_list ... ;
```

{Initially when A is DISPlayed, C would also be displayed, indirectly. If a function network were connected to A to change the level of detail to something other than 3, then the test in B would fail and C would not be displayed.}

## FORMAT

name := IF PHASE is **state** THEN [**name1**];

## DESCRIPTION

Refers to a structure if the **PHASE** attribute is in a specified state (ON or OFF). (See SET RATE and SET RATE EXTernal commands).

## PARAMETERS

**state** - Phase setting to be tested (ON or OFF).

**name1** - Structure to be conditionally referenced.

## DEFAULT

If there is no SET RATE node or SET RATE EXTernal node higher in the display tree, the PHASE attribute will always be OFF.

## DISPLAY TREE NODE CREATED

IF PHASE operation node (conditional connection between two structures).

## INPUTS FOR UPDATING NODE

None.

## EXAMPLE

```
A:= SET RATE 10 15 THEN B;

B:= IF PHASE is ON THEN C;

C:= VECtor_list ... ;
```

{If A is DISPlayed, C will also be displayed for 10 refresh frames and not DISPlayed for 15 refresh frames repetitively.}

## FORMAT

```
name := ILLUMINATION x,y,z [COLOR h [,s [,i]]] [AMBIENT a];
```

## DESCRIPTION

Specifies light sources for shaded images created with the PS 340. An unlimited number of light sources may be specified. This.command is only used with the PS 340. For a detailed explanation of defining and interacting with shaded images, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2.

## PARAMETERS

x,y,z - A vector from the origin pointing towards the light source.

h - A real number specifying the hue in degrees around the color wheel. Pure blue is 0 and 360, pure red is 120, and pure green is 240.

s - A real number specifying saturation. No saturation (gray) is 0 and full saturation (full toned colors) is 1.

i - A real number specifying intensity. No intensity (black) is 0, full intensity (white) is 1.

a - A real number which controls the contribution of a light source to the ambient light. Increasing **a** for a light source increases its contribution to the ambient light.

## DEFAULTS

If no ILLUMINATION command is used, a default white light at (0,0,-1) with an ambient proportion of 1.0 is assumed. If intensity and saturation are not specified, they default to 1. If only hue and saturation are specified, intensity defaults to 1. The default for ambient proportion is 1.

(continued)

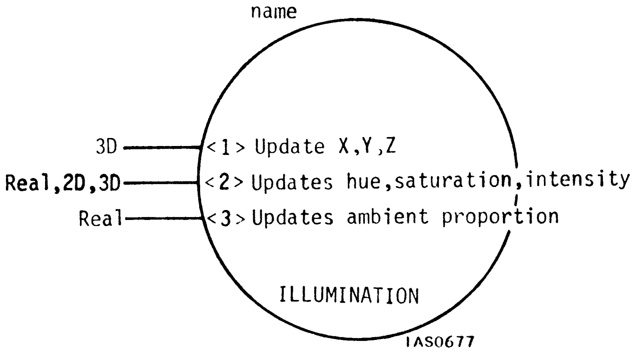
## NOTES

1. Illumination nodes may be placed anywhere in a display tree, allowing lights to be stationary or to rotate with the object, or both.
2. An unlimited number of light sources are valid for smooth-shaded renderings, but only the last illumination node encountered is used in creating flat-shaded renderings.
3. Light-sources are not used in wash-shaded (area-filled) images.

## DISPLAY TREE NODE CREATED

Illumination operate node.

## INPUTS FOR UPDATING NODE


name
3D
Real,2D,3D
Real
<1> Update X,Y,Z
<2> Updates hue,saturation,intensity
<3> Updates ambient proportion
ILLUMINATION
IAS0677

## NOTES ON INPUTS

A real number sent to input <1> changes only the hue. In this case, saturation and intensity default to 1. You cannot change just one value and retain the remaining values. Unless a 3D vector is sent, the default values are assumed for the variables not specified.

## EXAMPLE

```
Light := ILLUMINATION 1,1,-1 COLOR 180;
```

{This creates a node which defines a yellow light over the right shoulder. Since saturation and intensity are not specified, the defaults s = 1 and i = 1 are assumed. The ambient proportion defaults to 1.}

## FORMAT

INCLude name1 IN name2;

## DESCRIPTION

Used to include (instance) another named entity (name1) under a named instance node in a display tree (name2).

## PARAMETERS

name1 - Structure to be included under instance node name2.

name2 - Name of the instance node to include name1.

## DISPLAY TREE NODE CREATED

None. This is an immediate action command which modifies an existing instance node in a display tree.

## EXAMPLE

MAP:= INSTance of CANADA, SOUTH_AMERICA, UNITED_STATES;

INCLude MEXICO IN MAP;

{This would result in the instance node called MAP also pointing at MEXICO.}

## FORMAT

```
name := INCRement LEVel_of_detail[APPLied to name1];
```

## DESCRIPTION

Increments the current level of detail by 1 when **name** is being traversed.

## PARAMETERS

**name1** – Node to be affected by the increased level of detail.

## NOTE

There is really only one global level of detail; this command only changes the value of the level of detail while the named node and nodes below it in the display tree are being traversed.

## DISPLAY TREE NODE CREATED

INCREMENT LEVEL_OF_DETAIL operation node.

## INPUTS FOR UPDATING NODE

None.

(continued)

## EXAMPLE

```
A:= INCRement LEVel_of_detail THEN B;

B:= INSTance of C, D;

C:= IF LEVel_of_detail = 1 THEN E;

D:= IF LEVel_of_detail = 2 THEN F;
```

{If A were DISPlayed, E would also be displayed but not F. Since the default level of detail is 0, A will change the level of detail to 1, so the test in C will pass to E, while the test in D will fail and F will not be traversed.}

## GENERAL - Initialization

### FORMAT

INITialize [option];

### DESCRIPTION

INITialize (without specifying an option) restores the PS 300 to its initial state in which:

- No user-defined names exist.
- No user-defined units exist.
- No user-created display trees exist.
- No user-defined function connections exist.
- No structures are being displayed.

You may also initialize any of the above areas selectively (without initializing others) by following INITialize with the appropriate keyword for the area to be initialized.

The INITialize command also automatically executes the OPTIMIZE MEMORY command to collect any contiguous free blocks of memory into single blocks.

### PARAMETERS

option - Any of the following:

CONNections - Breaks all user-defined function connections.

DISPlay - Removes all structures from the display list.

NAMES - Clears the name dictionary of all structures and Function Instance names.

UNITS - Clears all user-defined units.

(continued)

NOTES

1. An INITialize command is specific to a command interpreter. It only affects the structures which were established by the same command interpreter as the initialization command itself. For example, structures created through the host line can be removed with an INITialize from the host, but not by an INITialize from the PS 300 keyboard.

2. The INITialize command blanks every object being displayed whether the object was created from the host or locally.

## FORMAT

name := INSTance of name1[,name2...,namen];

## DESCRIPTION

Groups one or more structures under a single named instance node.

## PARAMETERS

name1...namen - Structures to be grouped.

## DISPLAY TREE NODE CREATED

An instance node with pointers to each of the structures referenced (name1...namen).

## INPUTS FOR UPDATING NODE

None; however the INCLude and REMove commands can be used to modify the instance node.

## EXAMPLE

A:= INSTance of B,C,D;

## FORMAT

```
name := LABELS  x,y [,z] 'string'
                      .
                      .
                  [xi,yi [,zi] 'string'];
```

## DESCRIPTION

The LABELS command, like CHARacters, defines character strings for display. However, a single LABELS command can define an indefinitely large number of character strings.

## PARAMETERS

x,y,z - Coordinates of the lower left-hand corner of the first character in the string.

string - Text string up to 240 characters in length.

## DEFAULT

If z is not specified, it is assumed to be 0.

## NOTES

1. A gain in display capacity is realized whenever two or more character strings are combined in a single LABELS command.

2. The smallest LABELS entity that can be picked is an entire string; a pick returns an index into the LABELS command's list of strings. Individual characters cannot be picked as they can with CHARacters.

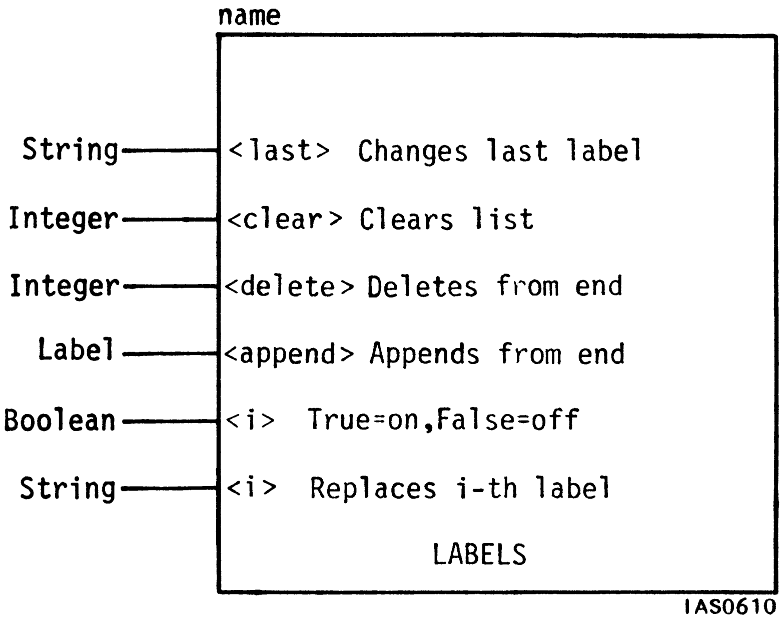
3. The commands SET CHARacters SCREEN_oriented/[FIXED] and SET CHARacters WORLD_oriented can be applied to LABELS in the same way they are applied to CHARacters.

4. You may SEND messages to a LABELS node as you can to a CHARacters node.

## DISPLAY TREE NODE CREATED

LABELS data node.

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

1. Sending an integer to <delete> of a LABELS node deletes that many strings from the end of the labels block. If the integer is as large as or larger than the number of strings in the block, then all strings are removed except the first. This is retained to keep the step size information, but display of that string is disabled.

2. Sending an integer to <clear> of a LABELS node deletes all labels except the first, which is retained for step size information, but is not displayed.

(continued)

NOTES ON INPUTS (continued)

3. The <append> input accepts only special "label" type messages that give both the string and the position to be appended. This data type is created by the F:LABEL function.

EXAMPLE

```
A:= LABELS 0,0 'FIRST LINE'
    0,-1.5 'SECOND LINE';
```

## FORMAT

```
name := LOOK AT ax,ay,az FROM fx,fy,fz
             [UP ux,uy,uz] [APPLied to name1];

name := LOOK FROM fx,fy,fz AT ax,ay,az
             [UP ux,uy,uz] [APPLied to name1];
```

## DESCRIPTION

This command, in conjunction with a windowing command (WINDOW, Field_Of_View, or EYE), fully specifies the portion of the data space that will be viewed, as well as the viewer's own orientation in the world coordinate system.

The LOOK AT...FROM clauses specify the viewer's position with respect to the object(s), while the optional UP clause specifies the screen "up" direction (analogous to adjusting the way the viewer's head is tilted).

LOOK creates a 4x3 transformation matrix which:

1. Translates the data base so that the FROM point is at the origin (0,0,0).

2. Rotates the data base so that the AT point is along the positive z axis at (0,0,D), where D = ||F-A||.

3. Rotates the data base so that the UP vector is in the YZ plane.

## PARAMETERS

ax,ay,az - Point being looked at, in world coordinates.

fx,fy,fz - Location of viewer's eye, in world coordinates.

ux,uy,uz - Vector indicating screen "up" direction.

name1 - Any structure.

(continued)

## DEFAULT

LOOK AT 0,0,1 FROM 0,0,0 UP 0,1,0;

## NOTES

1. To be implemented properly in a display tree, the LOOK node must follow one of the windowing nodes and may not precede any windowing node. (See note 1 for WINDOW.)

2. The UP vector indicates a direction only; its magnitude does not matter. For example, the two clauses UP 0,1,0 and UP 0,10,0 have exactly the same effect.

3. In determining FRONT and BACK boundary parameters for an associated windowing command (WINDOW, FIELD_Of_View, or EYE), remember that the LOOK command positions the AT point along the positive Z axis at 0,0,D where D equals the distance of the FROM point to the AT point. So, for example, if the FROM to AT distance is 13, if full depth cueing is desired, and the radius of the object is 1, then

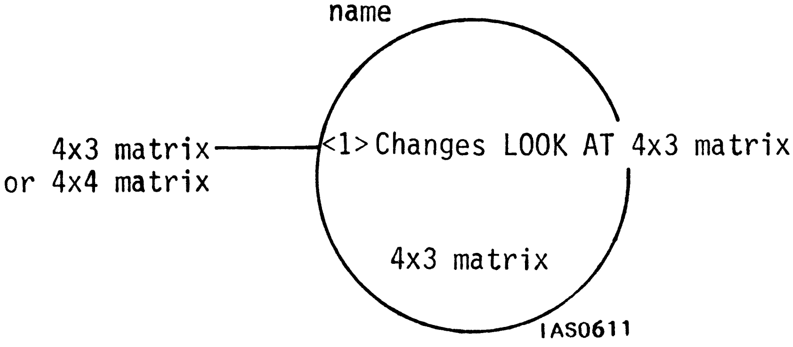
   FRONT boundary = 12
   BACK boundary = 14

   is used.

## DISPLAY TREE NODE CREATED

4x3 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

If a 4x4 matrix is input, the 4th column is ignored.

## ASSOCIATED FUNCTIONS

F:LOOKAT

## EXAMPLE

```
A:= BEGIN_Structure
     WINDOW X = -1:1 Y = -1:1
          FRONT boundary = 12
          BACK boundary = 14;
          LOOK AT 0,0,0 FROM 5,6.63,-10 THEN Sphere;
     END_Structure;
```

{If Sphere is defined with a radius of 1 about the origin, A would be a view of the Sphere from 5,6.63,-10, fully depth-cued. Note that the FROM to AT distance in the LOOK command is 13.}

## FORMAT

```
name := Matrix_2x2  m11,m12
                    m21,m22 [APPLied to name1];
```

## DESCRIPTION

Creates a 2x2 transformation matrix which applies to characters in the structure that follows (name1).

## PARAMETERS

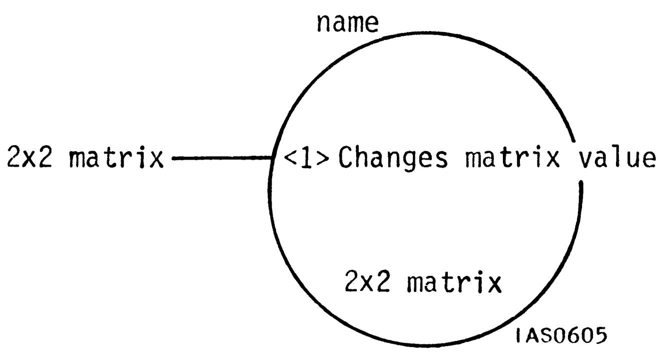
m11 - m22 - Elements of the 2x2 matrix.

name1 - Structure whose characters are to be transformed (any vector lists in the display tree are left unchanged).

## DISPLAY TREE NODE CREATED

2x2 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 2x2 matrix is legal.

## ASSOCIATED FUNCTIONS

F:MATRIX2, F:CSCALE, F:CROTATE

## EXAMPLE

```
A := MATRIX_2x2  1,0
                .5,1 THEN B;
```

{This creates a skewing matrix which is useful for italicizing text.}

## FORMAT

```
name := Matrix_3x3  m11,m12,m13
                    m21,m22,m23
                    m31,m32,m33 [APPLied to name1];
```

## DESCRIPTION

Creates a 3x3 transformation matrix which applies to the specified data (vector lists and/or characters).

## PARAMETERS

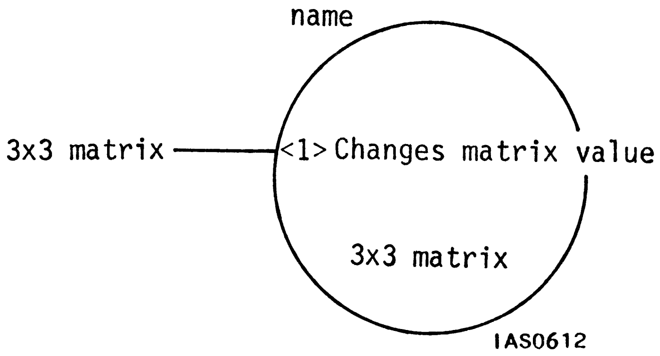
m11 - m33 – Elements of the 3x3 matrix to be created.

name1 – Structure to be transformed by the matrix.

## DISPLAY TREE NODE CREATED

3x3 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 3x3 matrix is legal (a rotation matrix, a scale matrix, etc.).

(continued)

## ASSOCIATED FUNCTIONS

F:MATRIX3, F:XROTATE, F:YROTATE,
F:ZROTATE, F:DXROTATE, F:DYROTATE,
F:DZROTATE, F:SCALE, F:DSCALE

## EXAMPLE

```
A := MATRIX_3x3 1,0,0
                0,1,0
                0,0,1 APPLied TO B;
```

{This creates an identity matrix.}

## FORMAT

```
name := Matrix_4x3  m11,m12,m13
                    m21,m22,m23
                    m31,m32,m33
                    m41,m42,m43 [APPLied to name1];
```

## DESCRIPTION

Creates a 4x3 transformation matrix which applies to the specified data (vector lists and/or characters).

## PARAMETERS

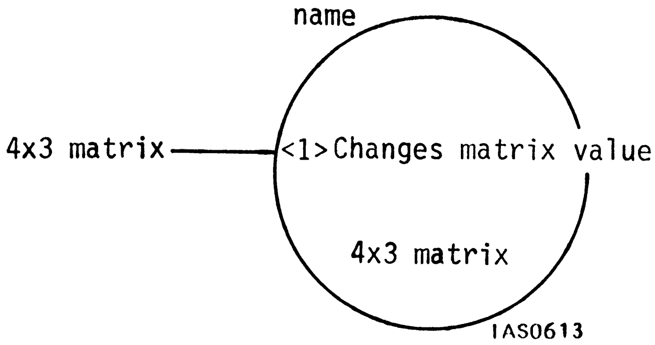
m11 - m43 - Elements of the 4x3 matrix to be created.

name1 - Structure to be transformed by the matrix.

## DISPLAY TREE NODE CREATED

4x3 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 4x3 matrix is legal (a rotation matrix, a scale matrix, etc.).

## ASSOCIATED FUNCTIONS

F:MATRIX4, F:XROTATE, F:YROTATE,
F:ZROTATE, F:DXROTATE, F:DYROTATE,
F:DZROTATE, F:SCALE, F:DSCALE

## EXAMPLE

```
A := MATRIX_4x3 1,0,0
                0,1,0
                0,0,1
                0,0,0 APPLied TO B;
```

## FORMAT

```
name := Matrix_4x4  m11,m12,m13,m14
                    m21,m22,m23,m24
                    m31,m32,m33,m34
                    m41,m42,m43,m44 [APPLied to name1];
```

## DESCRIPTION

Creates a 4x4 transformation matrix which applies to the specified data (vector lists and/or characters).

## PARAMETERS

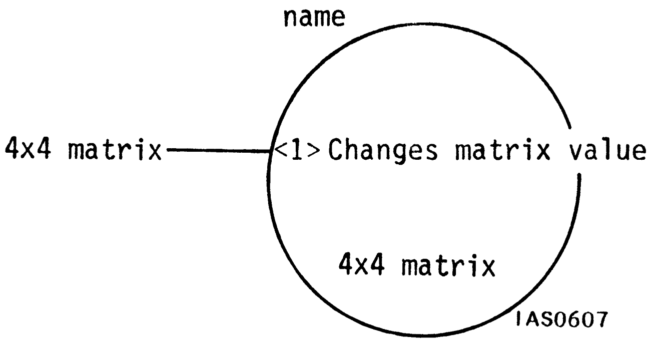
m11 - m44 – Elements of the 4x4 matrix to be created.

name1 – Structure to be transformed by the matrix.

## DISPLAY TREE NODE CREATED

4x4 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 4x4 matrix is legal (a rotation matrix, a scale matrix, etc.).

## ASSOCIATED FUNCTIONS

F:MATRIX4, F:XROTATE, F:YROTATE,
F:ZROTATE, F:DXROTATE, F:DYROTATE,
F:DZROTATE, F:SCALE, F:DSCALE

## EXAMPLE

```
A := MATRIX_4x4 1,0,0,0
                0,1,0,0
                0,0,1,0
                0,0,0,1 APPLied TO B;
```

{This creates an identity matrix.}

## FORMAT

```
name := display_data_structure_command;
```

## DESCRIPTION

Gives a name (address) to a node in a display tree so that it can be referenced explicitly.

## PARAMETERS

**name** - Any combination of alphanumeric characters up to 240. Must begin with an alpha character and can include $ and _.

**Display-data-structure_command** - All data structuring commands except the function instancing command (name := F:function_name).

## NOTES

1. All nodes in a display tree must be named (addressed) either directly, using this structure naming command, or indirectly, nesting a display data structure command within a BEGIN_Structure...END_Structure command.

2. Upper and lower-case letters can be used in names, but all letters are converted to upper-case. Thus turbine_blade, Turbine_Blade, and TURBINE_BLADE are equivalent names.

3. A null structure can be named using the **name := nil;** form of the command. If this command were used to redefine **name**, **name** would be kept in the name dictionary but the definition previously associated with **name** would be removed. FORGET **name** does just the opposite (see FORGET). DELETE **name** removes both the name and its definition (see DELETE).

## FORMAT

```
OPTIMIZE MEMORY;
```

## DESCRIPTION

An immediate action command which collects any contiguous free blocks of memory into single blocks.

## NOTES

1. If you are transmitting a large vector list from the host and you suspect that memory is being fragmented, enter this command before doing any operations.

2. This command is executed automatically whenever an INITialize command is entered.

## FORMAT

```
OPTIMIZE STRUCTURE;
      command;
      command;
         .
         .
END OPTIMIZE;
```

## DESCRIPTION

Places the PS 300 in, and removes it from, "optimization mode", during which certain elements of a display tree are created in a way that minimizes Display Processor traversal time.

## PARAMETERS

None.

## NOTES

1. Optimization mode is intended for application programs whose development is complete. Since optimization severely restricts the kinds of changes that may be made to a PS 300 display tree, it should not be used with programs whose structures may be changed.

2. To enter optimization mode for a developed application program, place the command

   ```
   OPTIMIZE STRUCTURE;
   ```

   at the beginning of the program (or portion of program) to be optimized, and place the command

   ```
   END OPTIMIZE;
   ```

   at the end.

NOTES (continued)

3. Optimization is not retroactive. The OPTIMIZE STRUCTURE command alone does not optimize any existing structures. On the other hand, structures created after the command is entered remain optimized even after END OPTIMIZE is entered, and even after legal changes are made to the structure.

4. The following changes may not be made to structures created or instanced during optimization mode:

   a. PREFIXes

   b. Redefinitions of data-definition commands (VECtor_list, CHARacters, LABELS, and polynomial and B-spline curves), regardless of whether or not the system is in optimization mode at the time of redefinition. Illegal changes to optimized structures have unpredictable effects on the display.

5. Among the types of structures for which optimization has an effect are INSTANCEs of multiple data-definition commands and BEGIN_S ... END_S structures containing only data-definition commands.

6. Optimization has no effect on a reference to a data-definition command which precedes the data-definition command itself.

7. OPTIMIZE STRUCTURE, like the INITialize command, affects only those structures created at the port at which the command is entered.

8. An INITialize command automatically performs an END OPTIMIZE.

## FORMAT

```
name := PATtern i [AROUND_corners][MATCH/NOMATCH]
            LENgth r;
```

## DESCRIPTION

Defines **name** to be a pattern. Patterns can be applied to existing vector lists (patterned and unpatterned) created by the WITH PATTERN, POLYNOMIAL, and BSPLINE commands. If curve commmands are used, the [AROUND_corners] option must be used.

## PARAMETERS

i - A series of up to 32 integers between 0 and 128 (delineated by spaces) indicating the relative lengths of alternating lines, spaces, lines, etc., in the pattern. The longer the series, the more complex the pattern of lines and spaces, which repeats every r units.

AROUND_corners - This indicates that patterning is to continue around each of the vectors in the vector list until the end of the list or a position vector is reached.

MATCH/NOMATCH - This indicates that the pattern length should be adjusted to make the pattern exactly match the end points of the vector or series of vectors being patterned. The default is MATCH.

r - The length over which i is defined and repeated.

## DISPLAY TREE NODE CREATED

None.

## FORMAT

PATTERN **name1** WITH **pattern;**

## DESCRIPTION

An immediate action command which applies a pattern to a vector list (**name1**).

## PARAMETERS

**pattern** – The pattern to be applied to **name1.** The pattern can be defined as either of the following.

**name** – A pattern created by the name := PATtern command

or

**i** [AROUND_corners] [MATCH/NOMATCH] LENgth r

where

**i** – A series of up to 32 integers between 0 and 128 delineated by spaces indicating the relative lengths of alternating lines, spaces, lines, etc., in the pattern. The longer the series, the more complex the pattern of lines and spaces, which repeats every r units.

**AROUND_corners** – This indicates that patterning is to continue around each of the vectors in the vector list until the end of the list or a position vector is reached.

**MATCH/NOMATCH** – This indicates that the pattern length should be adjusted to make the pattern exactly match the end points of the vector or series of vectors being patterned. The default is MATCH.

r – The length over which **i** is defined and repeated.

## DISPLAY TREE NODE CREATED

None.

## FORMAT

```
name := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [COPLANAR]
        POLYGon vertex ... vertex;
```

## DESCRIPTION

Allows you to define primitives as solids and surfaces. This command is only used with the PS 340. For a detailed explanation of defining and interacting with polygons, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2.

## PARAMETERS

WITH ATTRIBUTES - An option that assigns the attributes defined by **name1** for all polygons until superseded by another WITH ATTRIBUTES clause.

WITH OUTLINE - An option that specifies the color of the edges of a polygon on the color CSM display, or their intensity on a black and white display as a real number (**h**).

COPLANAR - Declares that the specified polygon and the one immediately preceding it have the same plane equation.

vertex - A vertex is defined as follows:

```
[ S ] x,y,z [ N x,y,z ]
```

where

S - indicates that the edge drawn between the previous vertex and this one represents a soft edge of the polygon. If the S specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft.

PARAMETERS (continued)

N - Indicates a normal to the surface with each vertex of the polygon. Normals are used only in smooth-shaded renderings. Normals must be specified for all vertices of a polygon or for none of them. If no normals are given for a polygon, they are defaulted to the same as the plane equation for the polygon.

x, y, z - are coordinates in a left-handed Cartesian system.

NOTES

1. A polygon declared to be coplanar must lie in the same plane as the previous polygon if correct renderings are to be obtained. The system does not check for this condition. Coplanar polygons may be defined without the coplanar specifier, unless outer and inner contours are being associated.

2. All members of a set of consecutive coplanar polygons are taken to have the same plane equation, that of the previous polygon not containing the coplanar option.

3. If coplanar is specified for the first polygon in a node, it has no effect.

4. If the N (normal) specifier is specified for a vertex in a polygon, it must be specified for all vertices in that polygon.

5. If the S (soft) specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft.

6. No more than 250 vertices per POLYGon may be specified.

7. The last defined vertex in the polygon is assumed to connect to the first defined vertex; that is, polygons are implicitly closed.

8. There is no syntactical limit for the number of POLYGon clauses in a group.

9. The ordering of vertices within each POLYGon has important consequences for rendering operations.

(continued)

## DISPLAY TREE NODE CREATED

Polygon data node.

## INPUTS FOR UPDATING NODE

None.

## FORMAT

```
name := POLYnomial[ORDER=i]
        [COEFFICIENTS=] xi,     yi,     zi
                        xi-1,   yi-1,   zi-1
                         .       .       .
                         .       .       .
                         .       .       .
                        0,      y0,     z0
        CHORDS=q;
```

## DESCRIPTION

Evaluates a parametric polynomial in the independent variable t over the interval [0,1]. This command allows the parametric description of many curve forms without the need to specify or transfer the coordinates of each constituent vector.

If the polynomial to be evaluated is called C, C is an i$^{th}$-order parametric polynomial in t such that:

$$C(t) = [\ x(t)\ y(t)\ z(t)\ ]$$

This polynomial may be expressed as the product of a vector (containing the various powers of t) and a coefficient matrix with three columns and i+1 rows:

$$C(t) = [ti\ \ ti\text{-}1\ \ldots\ t0] \begin{bmatrix} xi & yi & zi \\ xi\text{-}1 & yi\text{-}1 & zi\text{-}1 \\ . & . & . \\ . & . & . \\ . & . & . \\ x0 & y0 & z0 \end{bmatrix}$$

This coefficient matrix is what is specified in the polynomial command to represent the parametric polynomial C.

(continued)

## PARAMETERS

i - Optional specification of the order of the polynomial used to anticipate internal storage requirements.

xi, yi, zi - Coefficients of the polynomial.

q - The number of vectors to be created (0 < q < 32768).

## NOTES

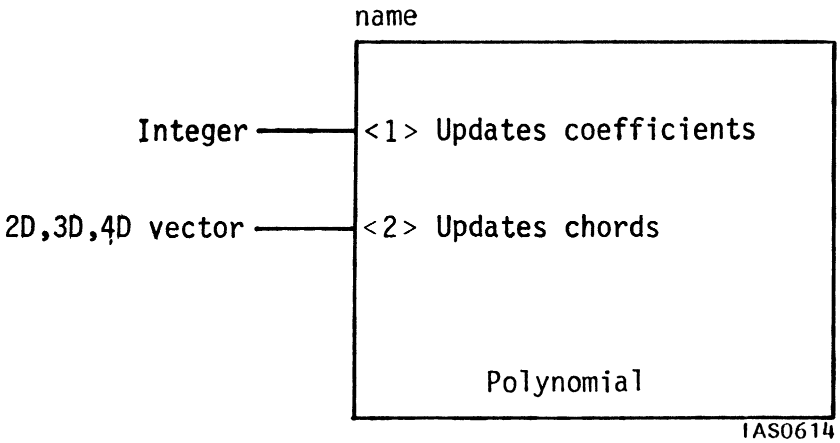
1. The interval [0,1] over which the polynomial in t is to be evaluated, is divided into q equal parts, so that C(t) is evaluated at t=0/q,1/q,2/q,...q/q. This causes the curve's constituent vectors to generally not be equal in length.

2. The polynomial's order is determined by the number of coefficient rows, and if the ORDER=i clause disagrees, it is ignored.

## DISPLAY TREE NODE CREATED

Polynomial vector list data node.

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

Sending a 2D vector to a 3D ploynomial node causes a default value of 0 to be used for z. If a 4D vector is sent to a 3D polynomial or a 3D or 4D vector is sent to a 2D polynomial, the w or z components are ignored.

## FORMAT

PREFIX **name** WITH **operation_command;**

## DESCRIPTION

Prefixes a named data node (**name**) with an operation node.

## PARAMETERS

**name** - A modeling primitive data node to be prefixed.

**operation_command** - Any PS 300 command that creates an operation node.

## NOTE

Any connections made to **namel** will be applied to the added prefix and not to the modeling primitive (i.e. **name** now points to the new operation node which points to the node that was previously **name**).

## DISPLAY TREE NODE CREATED

None. This is an immediate action command which just modifies an existing data node.

## EXAMPLE

A:= VECtor_list ...;

PREfix A WITH SCALE by .l;

{This will make A the name of a scale node pointing at a now unnamed vector list.}

## FORMAT

```
name := RATIonal BSpline ORDER=k
                    [OPEN/CLOSED] [NONPERIodic/PERIodic] [N=n]
                    [VERTICES =]  x1,y1,[z1],w
                                  x2,y2,[z2],w2
                                   .   .   .   .
                                   .   .   .   .
                                   .   .   .   .
                                  xn,yn,[zn],wn
                    [KNOTS = t1,t2,...,tj]
                    CHORDS =q;
```

## DESCRIPTION

Evaluates a rational B-spline curve, allowing the parametric description of the curve form without the need to specify or transfer the coordinates of each constituent vector.

The rational B-spline curve C is defined as:

$$C(t)=\frac{\sum_{i=1}^{n} w_i p_i N_{i,k}(t)}{\sum_{i=1}^{n} w_i N_{i,k}(t)}$$

where

$p_i$ - ith vertex of the B-spline's defining polygon

$N_{i,k}$ - ith B-spline blending function of order k

and

$w_i$ - weighting factor associated with each vertex (different weights determine the shape of the curve).

## DESCRIPTION (continued)

The parameter t of the curve and blending functions is defined over a sequence of knot intervals $t_1, t_2, \ldots, t_{n+k}$. Different knot sequences define different types of B-splines. Two common knot sequences are the uniform nonperiodic and uniform periodic knot sequences. A uniform nonperiodic B-spline is defined by the knot sequence:

$$t_j = \begin{array}{ll} 0 & (\text{for } j < k) \\ j-k & (\text{for } k < j < n) \\ n-k+1 & (\text{for } n < j < n+k) \end{array}$$

A uniform periodic B-spline is defined by the knot sequence:

$$t_j = j \ (\text{for } j < n+k)$$

The blending functions can be defined recursively as

$$N_{i,1}(t) = 1 \ (\text{if } t_i < t < t_{i+1}),\ 0 \text{ otherwise}$$

$$N_{i,k}(t) = \frac{(t-t_i)N_{i,k-1}(t)}{t_{i+k-1}-t_i} + \frac{(t_{i+k}-t)N_{i+1,k-1}(t)}{t_{i+k}-t_{i+1}}$$

The curve is evaluated at the points:

$$t = \frac{(1-i)t_k + i t_{j-k+1}}{q}$$

for $i=0,1,2,\ldots,q$.

## PARAMETERS

k - The order of the curve ($0 < k < 10$).

n - The number of vertices (used to anticipate storage requirements).

x1,y1,z1,w1...xn,yn,zn,wn - The vertices and weighting factor of the defining polygon of the curve. The z component is optional.

## PARAMETERS (continued)

tl,tl,...,tj - User specified knot sequence. Because closed B-splines are evaluated as open B-splines with duplicate vertices, the number of knots required is:

| | |
|---|---|
| n+k | for open B-splines |
| n+k+l | for closed nonperiodic B-splines |
| n+2K-l | for closed periodic B-splines |

The knots must also be nondecreasing.

q - The number of vectors to be created (0 < q < 32766).

## NOTES

1. **OPEN** or **CLOSED** is an option which describes the B-spline defining polygon. The default is **OPEN.** (Note that **CLOSED** merely describes the polygon, eliminating repetition of vertices. A full knot sequence, if specified, must be given.)

2. **NONPERIODIC** or **PERIODIC** is an option which specifies the default knot sequence. **NONPERIODIC** is the default for open B-splines; **PERIODIC** is the default for closed B-splines.

3. At least **k** vertices must be given, or the order **k** will be reduced accordingly.

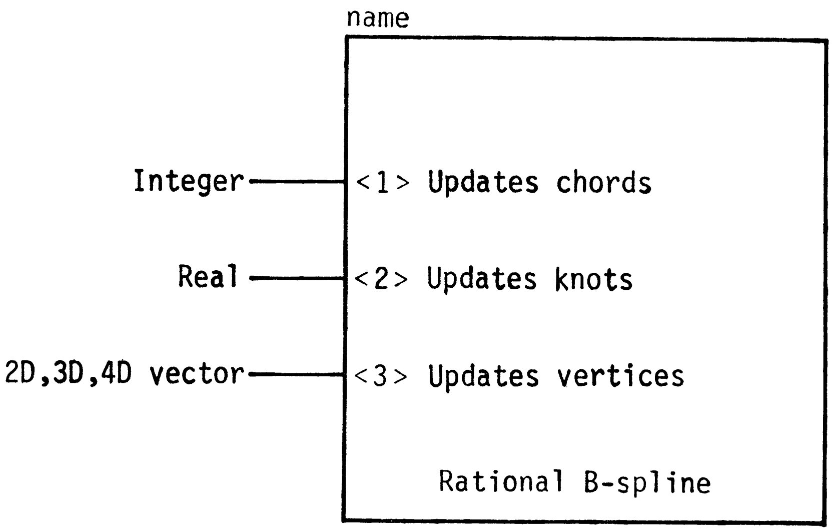
4. If all the weights of a rational B-spline are the same, tl.e curve is identical to the B-spline without the weights.

## DISPLAY TREE NODE CREATED

B-spline vector list data node.

(continued)

## INPUTS FOR UPDATING NODE



IAS0615

## NOTES ON INPUT

When a 2D vector is sent to a 3D rational B-spline, the default for *z* is 0 and for w is 1. The third component of 3D and 4D vectors is used as w in 2D rational B-splines.

## EXAMPLES

A third-order rational B-spline with defining polygon P1, P2, P3 defines a conic arc:

the arc is parabolic if w1=w2=w3
the arc is elliptic if w1=w3>w2
the arc is hyperbolic if w1=w3<w2

## FORMAT

```
name := RATional POLYnomial[ORDER=i]
          [COEFFICIENTS=] xi,   yi,   zi,   wi
                        xi-1, yi-1, zi-1, wi-1
                          .     .     .     .
                          .     .     .     .
                          .     .     .     .
                          x0,   y0,   z0,   w0
          CHORDS=q;
```

## DESCRIPTION

Evaluates a rational parametric polynomial in the independent variable t over the interval [0,1]. This command allows the parametric description of many curve forms without having to specify or transfer the coordinates of each constituent vector.

If the polynomial to be evaluated is called C, C is an $i^{th}$-order rational parametric polynomial in t such that:

$$C(t) = \begin{bmatrix} \dfrac{x(t)}{w(t)} & \dfrac{y(t)}{w(t)} & \dfrac{z(t)}{w(t)} \end{bmatrix}$$

This polynomial may be expressed as the product of a vector (containing the various powers of t) and a coefficient matrix with four columns and i+1 rows:

$$C(t) = [ti \ \ ti-1 \ \ldots \ t0] \begin{bmatrix} xi & yi & zi & wi \\ xi-1 & yi-1 & zi-1 & wi-1 \\ \vdots & \vdots & \vdots & \vdots \\ x0 & y0 & z0 & w0 \end{bmatrix}$$

This coefficient matrix is what is specified in the polynomial command to represent the rational parametric polynomial C.

(continued)

## PARAMETERS

i - Optional specification of the order of the polynomial used to anticipate internal storage requirements.

xi, yi, zi, wi - Coefficients of the polynomial.

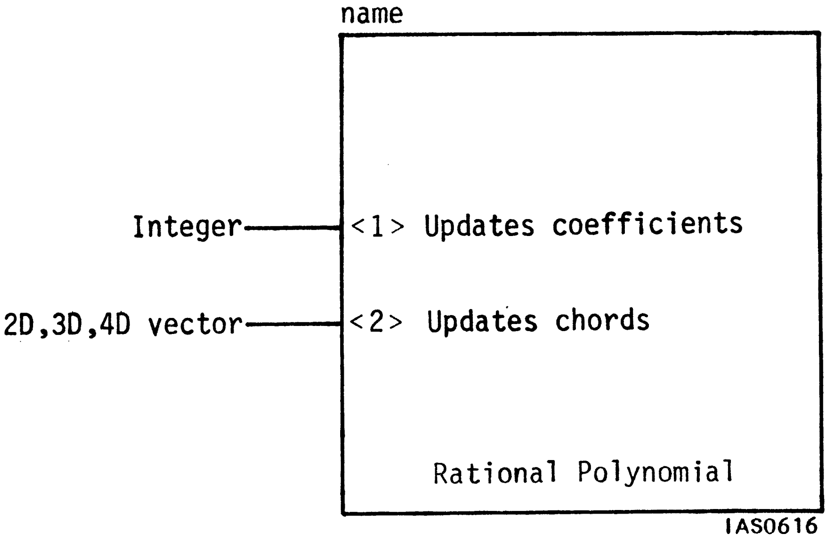
## NOTES

1. The interval [0,1] over which the polynomial in t is to be evaluated, is divided into q equal parts, so that C(t) is evaluated at t=0/q,1/q,2/q,...q/q.
2. Note that the curve's constituent vectors are not generally equal in length.
3. The polynomial's order is determined by the number of coefficient rows, and if the ORDER=i clause disagrees, it is ignored.

## DISPLAY TREE NODE CREATED

Rational polynomial vector list data node.

## INPUTS FOR UPDATING NODE



(continued)

NOTES ON INPUTS

Sending a 2D vector to a 3D ploynomial node causes a default value of 0 to be used for z and 1 for w. If a 4D vector is sent to a 3D polynomial or a 3D or 4D vector is sent to a 2D polynomial, the w or z and w components are ignored. The third component of 3D and 4D vectors is used as w in a 2D rational polynomial.

EXAMPLE

```
CIRCLE:= BEGIN_Structure

          RATional POLYnomial
             2,  0,  0,  2
            -2, -2,  0,  2
             0,  1,  0, -1
          CHORDS = 25;

          RATional POLYnomial
             2,  0,  0, -2
            -2, -2,  0,  2
             0,  1,  0, -1
          CHORDS = 25;

          END_Structure;

{This will create right and left semi-circles of radius 1.}
```

## FORMAT

```
name := REBOOT password;
```

## DESCRIPTION

Causes the PS 300 to reboot just as if it had been powered up, that is, it starts the confidence tests beginning with 'A'.

## PARAMETERS

password - System password set up by the PS 300 system manager.

## NOTES

1. If a password has been set up, an incorrect password will give an error message. If no password has been setup, any character string will cause the PS 300 to reboot.

2. REBOOT may be used inside a BEGIN_Structure ... END_Structure or outside.

## DISPLAY TREE NODE CREATED

None.

## FORMAT

REMove **name**;

## DESCRIPTION

Stops the display of **name**, that is, removes **name** from the display list.

## PARAMETERS

**name** - Any structure name.

## NOTE

Does not affect any structures in memory.

## FORMAT

REMove FOLLOWER of **name**;

## DESCRIPTION

Removes a previously placed follower of **name** (see FOLLOW WITH command).

## PARAMETERS

**name** – Structure that was previously modified with a **FOLLOW WITH** command.

## EXAMPLE

(Refer to the example given in the FOLLOW WITH command.)

REMove FOLLOWER of Shape.Rot;

{This command will restore the structure Shape to what it was originally (i.e. before the FOLLOW WITH command was given.)}

## FORMAT

REMove **name1** FROM **name2**;

## DESCRIPTION

Used to remove a named node (**name1**) from a named instance node (**name2**) in a display tree.

## PARAMETERS

**name1** - Node to be removed from instance node **name2**.

**name2** - Instance node that will no longer point to **name1**.

## DISPLAY TREE NODE CREATED

None. This is an immediate action command which just modifies an existing instance node.

## EXAMPLE

```
MAP:= INSTance CANADA, SOUTH_AMERICA, UNITED_STATES;

REMOVE SOUTH_AMERICA FROM MAP;

{This makes the instance of MAP point at CANADA and UNITED_STATES only.}
```

## FORMAT

REMove PREfix of **name**;

## DESCRIPTION

Removes a previously placed prefix (see PREFIX WITH command).

## PARAMETERS

**name** - Structure that was previously modified by a PREFIX WITH command.

## NOTE

This immediate action command restores **name** to what it was before being modified by a PREFIX WITH command.

## EXAMPLE

```
A:= VECtor_list ...;

PREfix A WITH SCALE by .1;

REMove PREfix of A;
```

{This will remove the previously PREfixed SCALE node, and A will once again be the name of the VECtor_list.}

## FORMAT

RESERVE_WORKING_STORAGE size;

## DESCRIPTION

Reserves a block of Mass Memory for sectioning plane, hidden-line removal, and backface removal renderings of solid objects defined as polygons. This command is used only with the PS 340.

## PARAMETERS

size - The number of bytes of Mass Memory that are reserved.

## NOTES

1. Renderings and saved renderings reside in mass memory along with the rest of the display structure. The original polygon is also stored in mass memory.

2. Each polygon of a solid object with four vertices will require approximately 150 bytes of reserve working storage. Memory needs will vary from figure to figure dependent upon the complexity of the object, the operations to be performed, and the view.

3. After one reserve-working-storage request is made, subsequent requests do not add to the original memory block -- they replace the original memory block.

4. If a contiguous block of memory cannot be allocated, no working storage is allocated and any previous storage is deallocated. If working storage is too small or has not been reserved, the rendering request is ignored and an error message is issued.

5. The best time to use RESERVE_WORKING_STORAGE is after booting, when large requests can be filled more easily. However, the command may be entered at any time.

NOTES (continued)

6. Typically, 200,000 to 400,000 bytes of working storage should be reserved at the beginning of a session.

7. A previously allocated block of memory is released prior to filling the request for a new block. Thus, a request for a smaller working storage area can always be fulfilled. However, because the working storage must be a contiguous block of memory, even slight increases in the working storage size may not be satisfied.

8. If working storage is too small or has not been reserved, the rendering request is ignored and an error message is issued.

## FORMAT

```
!RESET;
```

## DESCRIPTION

The !RESET command is used to get out of unended BEGIN's or BEGIN_STRUCTURE's when a problem occurs. (See also COMmand STATus.)

## FORMAT

name := ROTate in [axis] angle [APPLied to name1];

## DESCRIPTION

Rotates a structure (name1). Creates a 3x3 rotation matrix which rotates the specified data (vector lists and/or characters) about the designated axis, relative to the world coordinate system's origin. When you look in the positive direction of a given axis, positive angle values cause counterclockwise rotations (following the left-hand rule).

## PARAMETERS

axis - X, Y, or Z. If no axis is specified, the default is Z.

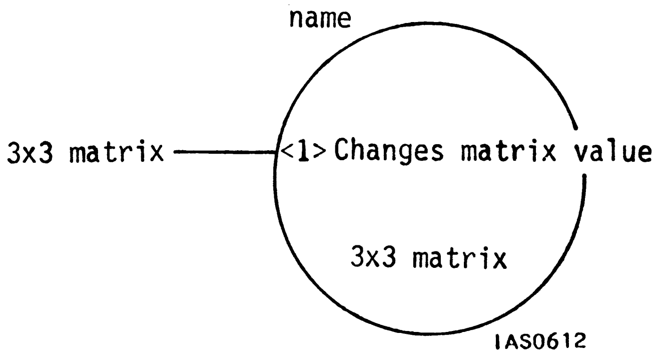
angle - Rotation angle in degrees (if no other units have been specified as default, and if no other units are explicitly specified in the ROTATE command).

name1 - Structure to be rotated.

## DISPLAY TREE NODE CREATED

3x3 matrix operation node.

## INPUTS FOR UPDATING NODE



(continued)

## NOTE ON INPUT

Any 3x3 matrix is legal (any rotation matrix, a scale matrix, a compound 3x3 matrix, etc.).

## ASSOCIATED FUNCTIONS

F:MATRIX3, F:XROTATE, F:YROTATE,
F:ZROTATE, F:DXROTATE, F:DYROTATE,
F:DZROTATE, F:SCALE, F:DSCALE

## EXAMPLE

```
A:= ROTate in X 45 THEN B;

B:= VECtor_list ... ;
```

## FORMAT

```
name := SCALE by s [APPLied to name1];
name := SCALE by sx,sy[,sz] [APPLied to name1];
```

## DESCRIPTION

Scales an object. Applies a uniform (s) or nonuniform (sx,sy,sz) 3x3 scale matrix transformation to the specified data (vector lists and/or characters).

## PARAMETERS

s - Uniform scaling factor (same along all axes).

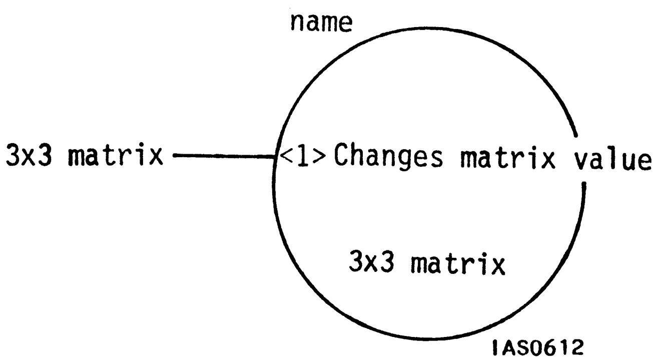
sx,sy,sz - Axial scaling factors. If sz is not specified, it is assumed to be 1 (no Z-scaling).

name1 - Object to be scaled.

## DISPLAY TREE NODE CREATED

3x3 matrix operation node.

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 3x3 matrix is legal (another scale matrix, a rotation matrix, etc.).

(continued)

## ASSOCIATED FUNCTIONS

F:MATRIX3, F:XROTATE, F:YROTATE,
F:ZROTATE, F:DXROTATE, F:DYROTATE,
F:DZROTATE, F:SCALE, F:DSCALE

## EXAMPLE

```
A:= SCALE by 5,2,3 THEN B;

B:= VECtor_list ... ;
```

## FORMAT

```
name := SECTioning_plane APPLied to name1;
```

## DESCRIPTION

Defines a sectioning plane, which is needed to produce a scetioned rendering of an object. This command is only used with the PS 340.

## PARAMETERS

**name1** - Either a POLYGon command or an ancestor of a POLYGon command.

## NOTES

1. Defining, displaying, and positioning a sectioning plane are the first steps in producing a sectioned rendering of an object. Hidden-line removal and backface removal do not require sectioning planes, but they can be used in conjunction with sectioned renderings.

2. The data which actually define a sectioning plane are contained in a POLYGon node; SECTioning_plane simply indicates that a given POLYGon represents a sectioning plane rather than an object to be rendered.

3. The sectioning plane is the plane in which a specified POLYGon lies. The polygon itself need not intersect the object to be sectioned, as long as some part of the plane does.

4. The sectioning plane is the plane containing the polygon defined by the first POLYGon clause of the first polygon node encountered by the Display Processor as it traverses the branch beneath a sectioning-plane node.

5. If the polygon node has more than one POLYGon, only the first polygon determines the sectioning plane. The other polygons have no effect on sectioning operations, but are displayed along with the defining polygon. This can be put to good use in designing an indicator which shows the side of the plane at which sectioning will remove (or preserve) polygon data.

(continued)

NOTES (continued)

6. A node may be a descendant of a sectioning-plane node if and only if it may be a descendant of a rendering operate node. Refer to the Notes on the SOLID_rendering command for permitted and prohibited descendant nodes.

7. If objects are to be sectioned, matrix-transformation nodes may be placed above the sectioning-plane node when and only when they are also ancestors of the objects' SOLID_RENDERING or SURFACE_RENDERING node(s). Failure to observe this rule results in bad renderings.

8. No SOLID_rendering or SURFACE_rendering operation node, whether below or above the sectioning-plane node, may be an ancestor of a sectioning plane's defining POLYGon. The PS 340 interprets such POLYGons as objects to be rendered rather than as sectioning-plane definitions, and issues a "Sectioning plane not found" message when a sectioning attempt is made. Other nodes which do not represent matrix viewing transformations, such as SET RATE and SET PLOTTER, may be placed either above or below the sectioning-plane node as needed.

9. Before an object can be sectioned, the sectioning-plane node must be part of a structure which is DISPlayed. If the plane's defining POLYGon is itself DISPlayed but its sectioning-plane node is not, no renderings can be created.

## DISPLAY TREE NODE CREATED

Sectioning-plane operation node.

## INPUTS FOR UPDATING NODE

None.

## FORMAT

SEND option TO <n>name1;

## DESCRIPTION

Sends a value to input n of Function Instance, node, or variable name1.

## PARAMETERS

option – The value to be sent. This can be any of the following forms:

i – A real number (with or without decimal point).

FIX(i) – Designates i to be an integer value (without decimal point).

V2D(i,j) – 2D vector.

V3D(i,j,k) – 3D vector.

V4D(i,j,k,l) – 4D vector.

M2D(a11,a12 a21,a22) – 2x2 matrix.

M3D(a11,a12,a13 a21,a22,a23 a31,a32,a33) – 3x3 matrix

M4D(a11,a12,a13,a14 a21,a22,a23,a24 a31,a32,a33,a34 a41,a42,a43, a44) 4x4 matrix

Boolean – TRUE or FALSE

'string' – A character string of one or more characters.

CHAR(m) – A single character whose decimal ASCII value is m.

P,L – Position or line.

VALUE(variable_name) – The value currently in variable_name, where variable_name is a previously declared PS 300 variable.

(continued)

EXAMPLE

```
TIMER:= F:CLCSECONDS;

SEND FIX(10) TO <1>TIMER;

{This puts an integer 10 on input 1 of TIMER.}
```

# SEND number * mode

## FORMAT

```
SEND number*mode TO <n>name1;
```

## DESCRIPTION

Sends to a vector list or labels node to change a specified number of vectors from position vectors to line vectors, or to turn a specified number of labels on or off.

## PARAMETERS

**number** – An integer specifying the number of vectors or labels.

**mode** – Either a **P** or **L**. For vector lists, P indicates a position vector and L indicates a line vector. For a labels block, P turns the label off, L turns it on.

n – An integer which identifies the first vector or label to receive the new specification.

**name1** – The destination vector list or labels node.

## FORMAT

SEND VL(name1) TO <i>name2;

## DESCRIPTION

Overwrites or appends vectors in vector lists or labels in label blocks.

## PARAMETERS

name1 – Name of vector list, character string, or label block to be sent.

name2 – Name of the destination VECtor_list or LABELS node.

i – An integer that specifies the first vector or first label to be replaced in name2 with vectors or labels in name1.

## NOTES

1. The parameter i can be replaced with last or append.
2. If i exceeds the number of vectors or labels in name2, the command will be ignored.

## FORMAT

name := SET CHARacters orientation [APPLied to name1];

## DESCRIPTION

Sets the type of screen orientation you want for displayed character strings.

## PARAMETERS

orientation - Three types of orientation may be set:

WORLD_oriented - Characters are transformed just like any part of the object containing them.

SCREEN_oriented - Characters are not affected by ROTate or SCALE transformations. Intensity and size of characters still vary with depth (Z-position).

SCREEN_oriented/FIXED - Characters are not affected by ROTate or SCALE transformations. They are always displayed with full size and intensity.

name1 - Structure affected by the SET CHARacters node.

## DEFAULT

SET CHARacters WORLD_oriented;

## DISPLAY TREE NODE CREATED

SET CHARacters operation node.

## INPUTS FOR UPDATING NODE

None.

## FORMAT

```
name := SET COLOR hue,sat [APPLied to name1];
```

## DESCRIPTION

Specifies the color of an object (**name1**).

## PARAMETERS

**hue** - A real number greater than or equal to 0 and less than 360, where:

0 = pure blue
120 = pure red
240 = pure green
360 = pure blue

**sat** - A real number from 0 to 1 where:

0 = no saturation (white)
1 = full saturation

**name1** - Structure to be colored.

## DEFAULT

The default setting for both **hue** and **sat** is 0.

## NOTE

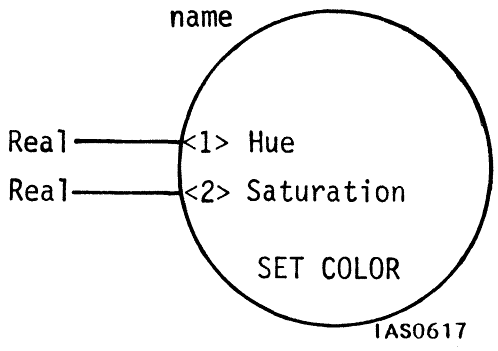
Zero saturation in any hue is white.

## DISPLAY TREE NODE CREATED

SET COLOR operation node.

(continued)

INPUTS FOR UPDATING NODE



EXAMPLE

```
A:= SET COLOR 240,1 THEN B;
B:= VECTOR_LIST ......;
```

{If A is displayed, the vector list described by B will be displayed in a pure green hue on a CSM.}

## FORMAT

name := SET COLOR BLENDing **sat** [APPLied to **name1**];

## DESCRIPTION

This command, used in conjunction with the COLOR option of the VECtor_list command, allows individual vector hue specifications. The ability to specify vector hues individually is called "color blending" because, if two adjacent vectors are of different hues, the hue of the line segment drawn between them is blended continuously between the endpoints. Vectors can only be color-blended when the contrast is set to zero using the SET CONTrast command.

## PARAMETERS

**sat** - A real number between 0 and 1, where 0 represents no color saturation (white) and 1 represents full color satuation.

**name1** - Either a VECtor_list command containing the COLOR option, or an ancestor of one or more such commands. **name1** may also be a VECtor_list command without the **COLOR** option (or ancestor thereof) provided that the SET COLOR command is applied to these lists as described in the notes that follow.

## NOTES

1. A color definition requires the specification of hue, saturation, and intensity. With SET COLOR BLENDing:

   hue values are specified individually for each vector

   a single saturation value is specified for all vectors

   intensity is always full (color-blended vector lists cannot be depth-cued), and the z values of vectors affect the color rather than the intensity. (For this reason, 2D vector lists are generally the most useful in color-blending.)

NOTES (continued)

2. Note that the "I=" clause of the VECtor_list command is not among the factors that determine a color-blended vector's intensity. Refer to the VECtor_list command for further details.

3. On systems lacking either a 2K ACP or a CSM Calligraphic Display, the SET COLOR BLENDing command is accepted but has no effect.

4. With color blended vector lists, the SET CONTrast command must be used to set the contrast of the PS 300 display to zero. If contrast is not set to zero, all color-blended vectors will appear blue.

## FORMAT

name := SET conditional_BIT n switch [APPLied to name1];

## DESCRIPTION

Alters one of the 15 global conditional bits temporarily, during the traversal of a branch of a display tree. These temporary settings may be tested further down within the display tree, possibly allowing conditioned reference to other structures (see IF conditional_BIT command). When traversal of the branch is complete, the bits are restored to their previous values.

## PARAMETERS

n – An integer from 0 to 14, corresponding to the conditional bit to be set ON or OFF by the command (see Note 1 below).

switch – ON or OFF.

name – Structure to follow the conditional bit node.

## DEFAULT

All 15 conditional bits are initially set to OFF.

## NOTES

1. Although only one conditional bit can be set ON or OFF by this command, a function network could be tied in to this node to set any conditional bit ON or OFF.

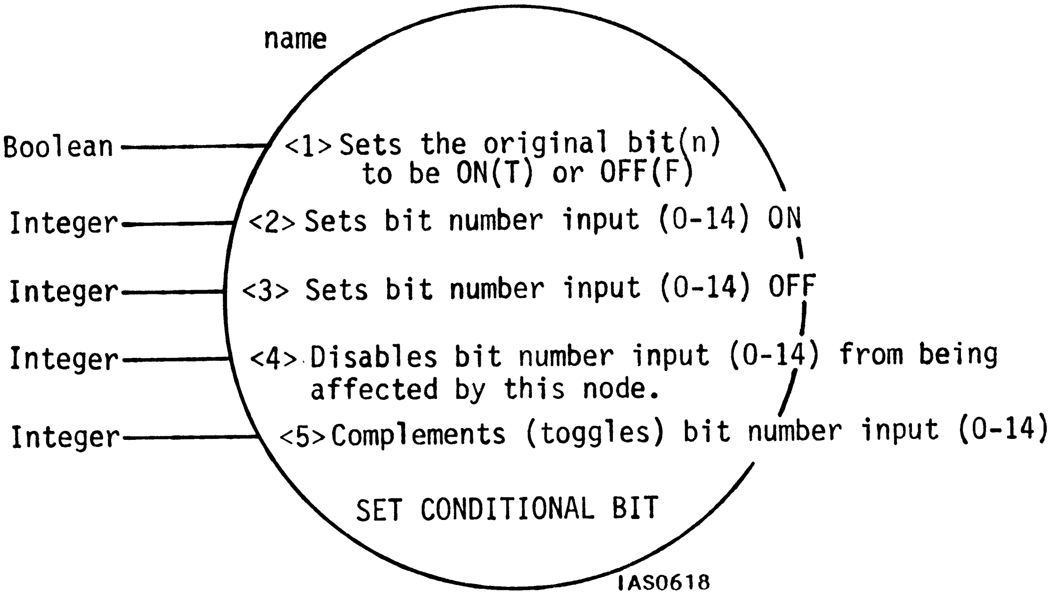
2. Note that there is really only one bank of 15 conditional bits and that this command only changes the values of these bits temporarily, while name1 is being traversed. However, descendants of name1 could also be SET conditional_BIT nodes. These are saved and restored as part of the state of the machine during the traversal of different branches of the display tree.

(continued)

## DISPLAY TREE NODE CREATED

SET conditional_BIT operation node.

## INPUTS FOR UPDATING NODE


name
Boolean
Integer
Integer
Integer
Integer
<1> Sets the original bit(n) to be ON(T) or OFF(F)
<2> Sets bit number input (0-14) ON
<3> Sets bit number input (0-14) OFF
<4> Disables bit number input (0-14) from being affected by this node.
<5> Complements (toggles) bit number input (0-14)
SET CONDITIONAL BIT
IAS0618

## EXAMPLE

A:= SET conditional_BIT 3 ON THEN B;

B:= IF conditional_BIT 3 is ON THEN C;

C:= IF conditional_BIT 6 is ON THEN D;

D:= VECtor_list ... ;

{A function network should be tied to A so that the state of any of the conditional BITs can be changed, not just the one that was initially set ON or OFF.}

## FORMAT

```
name := SET CONTrast to c [APPLied to name1];
```

## DESCRIPTION

Changes the contrast of the PS 300 display(s).

## PARAMETERS

c - A number from 0 to 1 (0 = lowest contrast, 1= highest contrast).

name1 - Structure using this contrast setting.

## DEFAULT

SET CONTrast to 1;

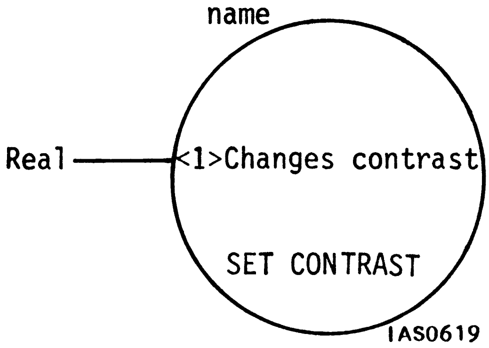
## NOTES

1. Setting contrast to 1 provides the highest contrast and thus the greatest perception of depth cueing (all else being equal).
2. Although any real value from 0 to 1 is legal for c, c is mapped to one of four values (0.,.33,.67,1.).
3. With color blended vector lists, SET CONTrast must be used to set the contrast of the PS 300 display to zero. If contrast is not set to zero, color-blended vectors will appear blue.

## DISPLAY TREE NODE CREATED

SET CONTrast operation node.

INPUTS FOR UPDATING NODES



EXAMPLE

A:= SET CONTrast to 0 THEN B;

B:= VECtor_list ... ;

{This is a minimum contrast setting.}

## FORMAT

name := SET CSM **switch** [APPLied to **name1**];

## DESCRIPTION

Allows you to specify one of two modes of operation for the CSM Calligraphic Display.

## PARAMETERS

**switch** – Two settings may be specified:

ON – This setting slows the Line Generator to half speed and provides extra brightness and precision (endpoint match and color convergence) in displayed data.

OFF – This is the default setting. It sets the Line Generator to ful speed, allowing for the maximum number of vectors to be displayed in a refresh cycle.

## NOTE

The following command should be added to the SITE.DAT file of any installation using a CSM Calligraphic Display:

```
SEND TRUE TO <1>CSM;
```

This command establishes SET CSM ON as the default mode for graphics display, and sets the Terminal Emulator and the Message Display line to be displayed in CSM mode.

## DEFAULT

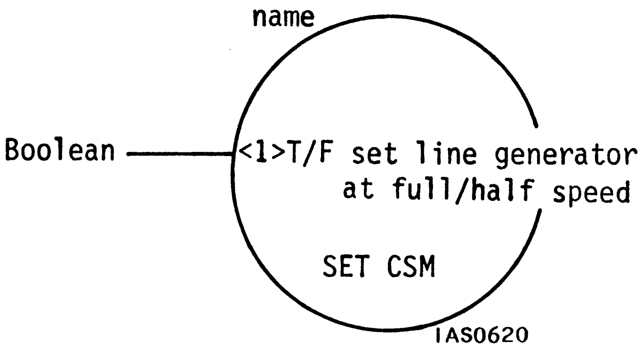
SET CSM OFF;

## DISPLAY TREE NODE CREATED

SET CSM operation node.

(continued)

INPUTS FOR UPDATING NODE



## FORMAT

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

## DESCRIPTION

Enables/disables Z-plane (depth) clipping.

## PARAMETERS

switch - ON or OFF.

name1 - Structure affected.

## DEFAULT

SET DEPTH_CLipping OFF;

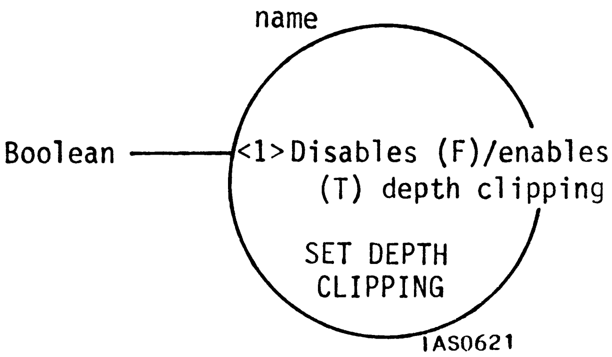
## NOTE

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.

## DISPLAY TREE NODE CREATED

SET DEPTH_CLipping operation node.

## INPUTS FOR UPDATING NODES



(continued)

EXAMPLE

```
A:= SET DEPTH_CLipping ON THEN B;

B:= ... ;

{This enables Z clipping.}
```

## FORMAT

```
name := SET DISPlays ALL switch [APPLied to name1];
name := SET DISPlay n[,m...] switch [APPLied to name1];
```

## DESCRIPTION

Specifies the scope(s) which are to receive display information.

## PARAMETERS

switch - ON or OFF.

n[,m...] - 0,1,2,3. Numeric designation for PS 300 scopes.

name1 - Structure to be displayed.

## DEFAULT

SET DISPLAYS ALL ON;

## NOTE

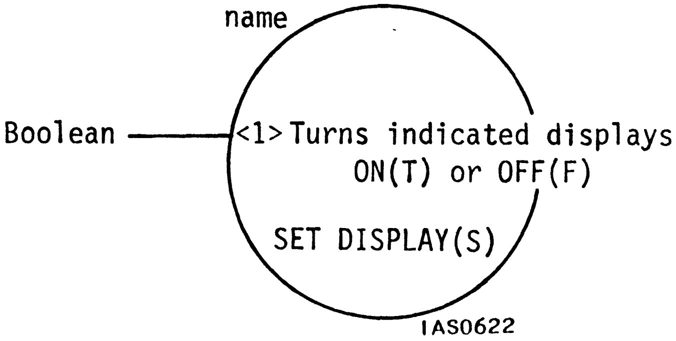
1. The ALL version of the command only refers to those scopes that have already been explicitly specified by a previous SET DISPlay command.

2. Scope numbers correspond to the hardware configuration (e.g., Scope 0 is the scope number when there is just one scope in the system).

## DISPLAY TREE NODE CREATED

SET DISPlay(s) operation node.

INPUTS FOR UPDATING NODES



EXAMPLE

A:= SET DISPlay 1 ON THEN B;

B:= VECtor_list ... ;

{This channels B to be displayed on scope 1.}

## FORMAT

name := SET INTENsity **switch imin:imax** [APPLied to **name1**];

## DESCRIPTION

Specifies intensity variation for depth cueing, and may be used to override the intensity specification associated with the VIEWPORT command or previous SET INTENsity commands.

## PARAMETERS

**switch** - Two settings may be specified: **ON** and **OFF**. The default setting is **ON**, which means enable the effect of this node in the display tree. **OFF** means disable the effect.

**imin** - A real number ranging from 0.0 to 1.0, **imin** represents the dimmest intensity setting.

**imax** - A real number ranging from 0.0 to 1.0, **imax** represents the brightest intensity setting.

**name1** - Structure to be affected.

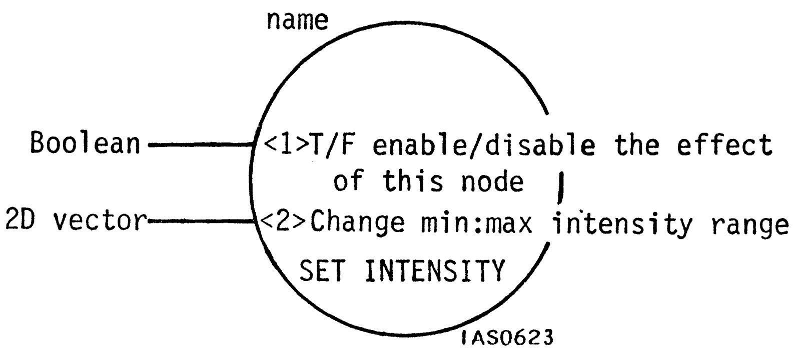
## NOTE

The last SET INTENsity node that is **ON** in a display tree determines the intensity range.

## DISPLAY TREE NODE CREATED

SET INTENsity operation node.

INPUTS FOR UPDATING NODE


name
Boolean
<1>T/F enable/disable the effect of this node
2D vector
<2>Change min:max intensity range
SET INTENSITY
IAS0623

## FORMAT

name := SET LEVel_of_detail to **n** [APPLied to **name1**];

## DESCRIPTION

Alters a global level of detail value temporarily, during the traversal of a specified branch of a display tree. These temporary settings may be tested further down within the display tree, possibly allowing conditioned reference to other structures (see IF LEVel_of detail command). When traversal of the branch is complete, the level of detail is restored to its original value.

## PARAMETERS

**n** - An integer from 0 to 32767 indicating the level of detail value.

**name** := Structure to be affected by the level of detail.

## DEFAULT

The level of detail is initially 0.

## NOTE

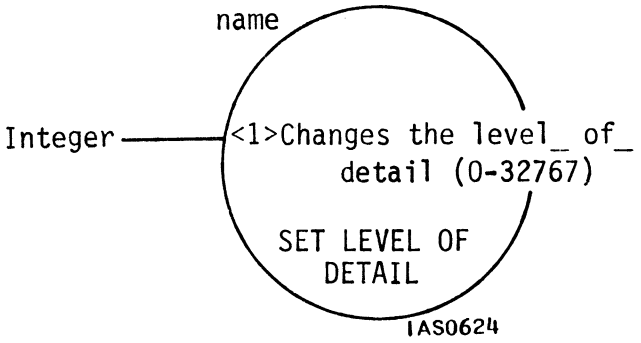
There is really only one global level of detail value; this command only changes the value of the level of detail temporarily, while the **name1** structure is being traversed.

## DISPLAY TREE NODE CREATED

SET LEVel_of_detail operation node.

(continued)

## INPUTS FOR UPDATING NODE



## EXAMPLE

```
A:= SET LEVel_of_detail to 2 THEN B;

B:= IF LEVel_of_detail = 2 THEN C;

C:= ... ;
```

{A function network should be tied to A to change the level of detail for conditional referencing of C.}

## FORMAT

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

## DESCRIPTION

Enables or disables picking for a specified structure.

## PARAMETERS

switch - ON or OFF for enabling or disabling picking.

name1 - Structure to be affected.

## NOTE

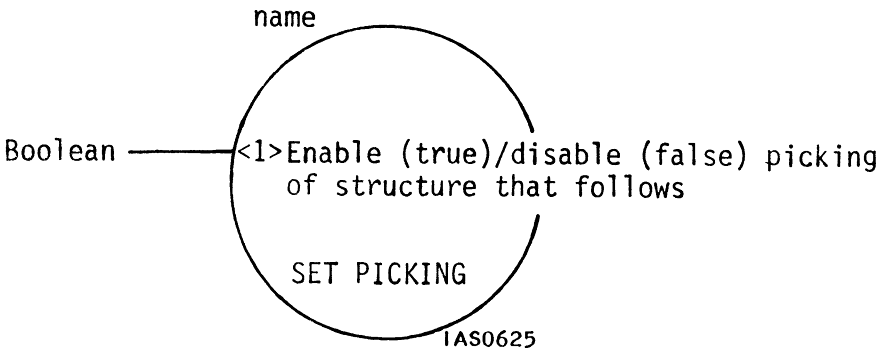
1. There must also be a SET PICKing IDentifier node in the structure to be pickable for picking to be reported.

2. See also SET PICKing LOCation and SET PICKing IDentifier.

## DISPLAY TREE NODE CREATED

SET PICKING operation node (information to enable/disable hardware picking).

## INPUTS FOR UPDATING NODE


name
Boolean
<1>Enable (true)/disable (false) picking of structure that follows
SET PICKING
1AS0625

(continued)

## EXAMPLE

```
A:= SET PICKing OFF THEN B;

B:= ... ;
```

{A function network should be tied to A to SET PICKing ON when needed in order to make structure B pickable.}

## FORMAT

name := SET PICKing IDentifier = **id_name** [APPLied to **name1**];

## DESCRIPTION

Specifies textual information that will be reported back if a pick occurs further down in the structure **name1**. Nested pick identifier names are all reported, separated by commas.

## PARAMETERS

**id_name** – Text that will be reported if a pick occurs anywhere within the structure **name1**. This must be a legal PS 300 name.

**name1** – Structure to which the pick ID applies.

## NOTES

1. At least one pick ID must precede any pickable entity for picking to be reported.

2. **id_name** cannot be updated by a function network.

## DISPLAY TREE NODE CREATED

SET PICKing IDentifier operation node.

## INPUTS FOR UPDATING NODE

None.

(continued)

## EXAMPLE

```
A:= SET PICKing OFF THEN B;

B:= SET PICKing IDentifier = structure_C THEN C;

C:= VECtor_list ... ;
```

{If a vector in C is picked, the ID name reported in the pick list will be structure_C.}

# SET PICKING LOCATION

## FORMAT

```
name := SET PICKing LOCation = x,y size_x,size_y;
```

## DESCRIPTION

Specifies a retangular picking area at (x,y) within the current viewport. The rectangle is bounded by (x ≥ size_x) and (y ≥ size_y).

If an appropriate picking network is set up and a pick-sensitive vector list (vectors or dots) is drawn within the pick location, it will be reported as picked.

## PARAMETERS

x,y - The center of the pick location.

size_x,size_y - Offsets from the x,y center specifying the bounds of the picking rectangle (the rectangle bounds must be within ≥1 range).

## DEFAULTS

A default pick location is set up in the configuration file that is loaded when the system is booted. The x,y center is tied to the position of the data tablet stylus, and size_x,size_y are both set to .01, (i.e., a box whose dimensions are .02 on each side).

## NOTES

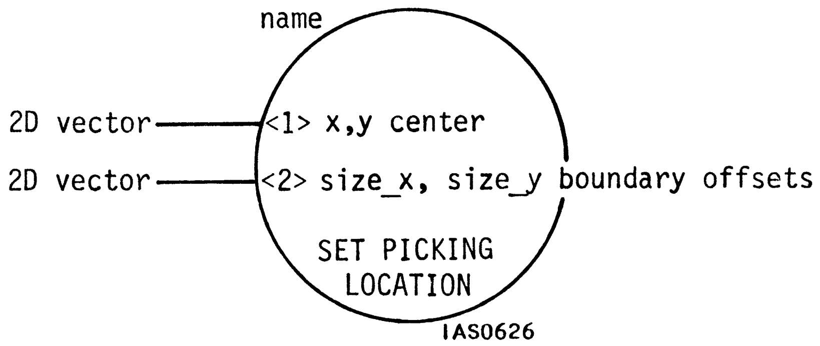
1. In most applications, the picking location needs to be moveable, so the x,y center is usually updated by a function network that specifies where the center should be.

2. The data tablet's x,y value is usually the source for specifying the pick location center.

## DISPLAY TREE NODE CREATED

SET PICKing LOCation operation node (information for hardware picking).

## INPUTS FOR UPDATING NODE



## ASSOCIATED FUNCTION

F:PICK

## EXAMPLE

PICK_LOCATION := SET PICKing LOCation = 0,0 .02,.02;

{This redefines the default picking area set up in the configuration file, making the picking area twice as large as the default.}

## FORMAT

name := SET PLOTTER **switch** [APPLied to **name1**];

## DESCRIPTION

Allows you to specify parts of a structure that are eligible for plotting.

## PARAMETERS

**switch** - Two settings may be used with the SET PLOTTER command:

ON - enables a structure to be plotted.

OFF - prevents a structure from being plotted.

**name1** - The structure in question.

## DEFAULT

SET PLOTTER ON;

## DISPLAY TREE NODE CREATED

SET PLOTTER operation node.

## INPUTS FOR UPDATING NODE

None.

## FORMAT

```
name := SET RATE phase_on phase_off [initial_state][delay]
                                    [APPLied to name1];
```

## DESCRIPTION

Temporarily alters two global duration values (phase_on and phase_off, in refresh frames) during the traversal of a specified branch of a display tree. These temporary settings may be tested further down within the display tree, possibly allowing conditioned reference to other structures (see IF PHASE command). When traversal of the branch is complete, the durations are restored to their original values.

## PARAMETERS

phase_on,phase_off - Integers designating the durations of the on and off phases, respectively, in refresh frames.

initial_state - ON or OFF, indicating the initial phase.

delay - Integer designating the number of refresh frames in the initial_state.

name1 - Structure to follow the SET RATE command.

## DEFAULT

The default phase is OFF and never changes unless a SET RATE node is encountered.

## NOTES

1. This structure attribute is useful for controlling blinking, the alternating display of two structures, the alternating display of a single structure in two different views (stereo), etc.

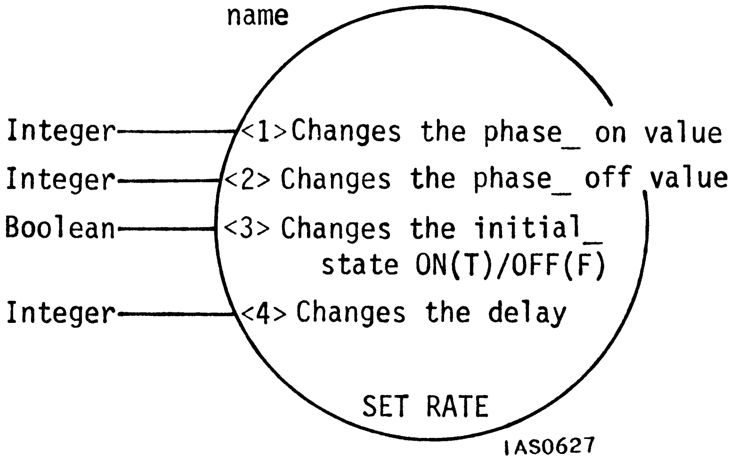
2. Note that there are only two rate values (phase_on, phase_off) and that this command only changes those values for the structure(s) that follow.

(continued)

## DISPLAY TREE NODE CREATED

SET RATE operation node.

## INPUTS FOR UPDATING NODE


name
Integer
<1>Changes the phase_ on value
Integer
<2> Changes the phase_ off value
Boolean
<3> Changes the initial_ state ON(T)/OFF(F)
Integer
<4> Changes the delay
SET RATE
IAS0627

## EXAMPLE

```
A:= BEGIN_Structure
rate:=    SET RATE 10 100;
          IF PHASE is ON THEN B;
     END_Structure;

B:= VECtor_list ... ;
```

{If A is DISPlayed, then vector list B will be displayed for 10 frames and not displayed for 100 frames repetitively.}

## FORMAT

name := SET RATE EXTernal [APPLied to name1];

## DESCRIPTION

Sets up a structure that can be used to alter the PHASE attribute via an external source, such as a function network or a message from the host computer. This PHASE attribute can be tested further down within the display tree, allowing conditional references to other structures (see IF PHASE command). See also the SET RATE command which alters the PHASE attribute based on refresh cycles.

## PARAMETERS

name1 - Structure to follow the SET RATE EXTernal command.

## DEFAULT

The default phase is ON when a SET RATE EXTernal node is encountered.

## NOTES

1. The PHASE attribute is changed by sending a Boolean value to input 1 of SET RATE EXTernal node.

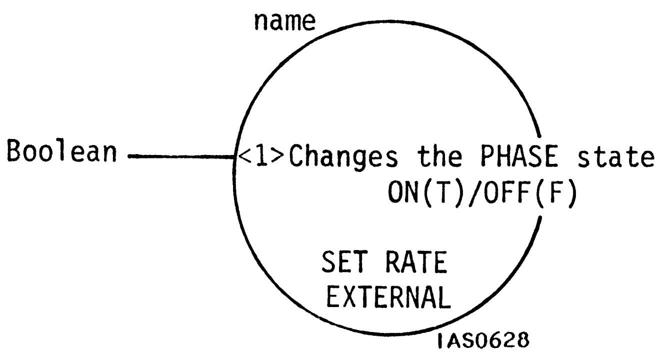
2. See also notes for SET RATE command.

## DISPLAY TREE NODE CREATED

SET RATE EXTernal operation node.

(continued)

## INPUTS FOR UPDATING NODE



## EXAMPLE

```
A:= BEGIN_Structure
rate:=    SET RATE EXTernal;
          IF PHASE is ON THEN B;
     END_Structure;

B:= VECtor_list ... ;
```

{A function network should be connected to A.rate to set the PHASE ON and OFF in order to conditionally display vector list B.}

## FORMAT

```
SETUP CNESS queue_type <i>name;
```

## DESCRIPTION

Allows you to specify whether or not an input queue to a function instance is to be a constant queue.

## PARAMETERS

**queue_type** - TRUE sets the queue type to constant, FALSE sets it to active.

**name** - Most intrinsic function names, except those listed in the notes.

## NOTES

1. This feature should only be used when a function is first instanced. Input queues should not be changed between active and constant after the function has started processing data.

2. The SETUP CNESS command can be used for all intrinsic functions except the following.

   F:LINEEDITOR, F:CLSECONDS, F:CLFRAMES, F:CLTICKS,
   F:BOOLEAN_CHOOSE, F:INPUTS_CHOOSE, F:PASSTHRU,
   F:XFORMDATA

3. Functions which specify their queue characteristics by their name, e.g., F:ADDC, will continue to be instanced with their default active and constant queues.

## FORMAT

name := SOLID_rendering APPLied to **name1**;

## DESCRIPTION

Declares a POLYGon object to be a solid and marks the object so that rendering operations can be performed on it. This command creates a rendering node. It is used exclusively with the PS 340.

## PARAMETERS

**name1** - Either a POLYGon node or an ancestor of one or more POLYGon nodes.

## NOTES

1. If non-POLYGon data nodes (such as VECtor_list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in **name1**, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes.

2. IF and SET CONDITIONAL BIT, IF and SET LEVEL_OF_DETAIL, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operate nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result. (Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings.) The transformation nodes mentioned bove may also be placed above the rendering node.

NOTES (continued)

4. The five nodes WINDOW, VIEWPORT, EYE, Field_Of_View, and MATRIX_4X4 should NOT, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWPORT, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiply-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR, SET PLOTTER) not mentioned above, are ignored by rendering operations when placed beneath a rendering node. Such nodes must therefore be placed above a rendering node if they are to have their customary effects on renderings. Data nodes other than POLYGON are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPLAYed. If the object itself is DISPLAYed but its rendering node is not, no renderings can be created.

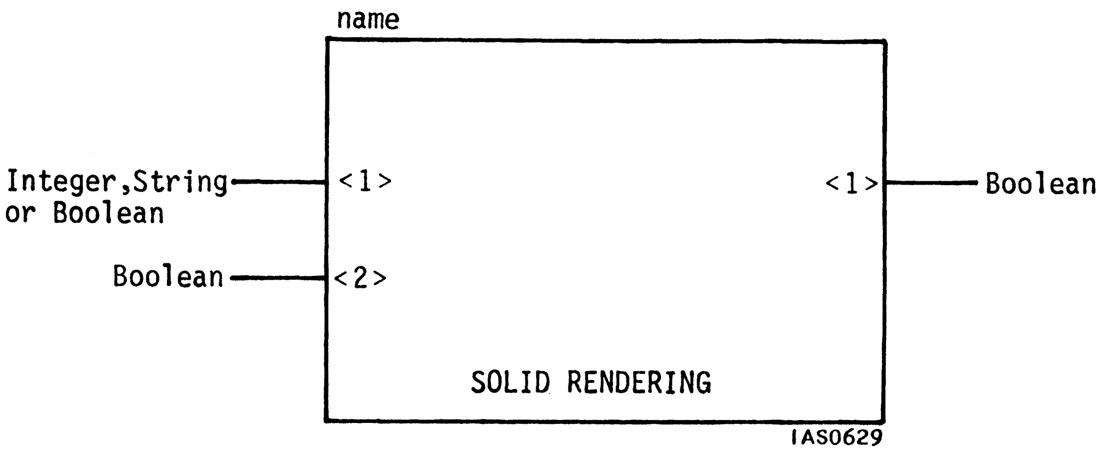
8. Any input to input<1> of a rendering node causes an output. Inputs sent to input<2> will not cause an output to be sent. If output<1> has not been connected, and an integer, string, or Boolean is sent to input<1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

DISPLAY TREE NODE CREATED

Rendering operation node.

(continued)

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object.
1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only).
2: Creates and displays a sectioned rendering.
3: Creates and displays a rendering using backface removal (solid only).
4: Creates and displays a rendering using hidden-line removal.
5: Generates a wash-shaded image on the raster display.
6: Generates a flat-shaded image on the raster display.
7: Generates a smooth-shaded image on the raster display.

String: Causes the current rendering to be saved under the name given in the string.
False: Sets the original view. The original descendant structure of the rendering operate node is displayed.
True: Sets the rendered view. The rendered view of the original descendent structure of the rendering operate node.

Input <2>

True: Declares the object to be a solid.
False: Declares the object to be a surface.

Output <1>

True: Rendering is displayed
False: Rendering is not displayed

## FORMAT

name := STANdard FONT [APPLied to name1];

## DESCRIPTION

Establishes the standard PS 300 95-character font as the working font.

## PARAMETERS

name1 - Structure to use the standard font.

## DEFAULT

If no other font is specified, the STANdard FONT is the default font.

## NOTE

This command is necessary only if the STANdard FONT is to be used in a display tree that uses another font higher in the same structure.

## DISPLAY TREE NODE CREATED

Character font pointer node.

(continued)

EXAMPLE

```
SLANT := BEGIN_Font
          (character definitions)
END_Font;

A := BEGIN_Structure
     character FONT SLANT;
     CHARacters 'HERE';
     STANdard FONT;
     CHARacters 0,-2 'HERE';
END_Structure;

DISPlay A;
```

{'HERE' at 0,0 will be in the SLANT font 'HERE' at 0,2 will be in the STANDARD font.}

## FORMAT

```
STORE option IN name1;
```

## DESCRIPTION

Sends a value to input <1> of Function Instance, node, or variable name1.

## PARAMETERS

option – See SEND command.

name1 – Function Instance name, node name, or variable name to receive value on input <1>.

## NOTE

This command is another way of doing a special case of the SEND command. It is synonymous with: SEND option TO <1>name1;

## EXAMPLE

```
TIMER:= F:CLCSECONDS;

STORE FIX(10) IN TIMER;

{This is equivalent to: SEND FIX(10) TO <1>TIMER;}
```

## FORMAT

name := SURFACE_rendering APPLied to name1;

## DESCRIPTION

Declares a POLYGon object to be a surface and marks the object so that rendering operations can be performed on it. This command creates a rendering node. It is used exclusively with the PS 340.

## PARAMETERS

**name1** - Either a POLYGon node or an ancestor of one or more POLYGon nodes.

## NOTES

1. If non-POLYGon data nodes (such as VECtor_list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in **name1**, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes.

2. IF and SET CONDITIONAL BIT, IF and SET LEVEL_OF_DETAIL, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operate nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result. (Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings.) The transformation nodes mentioned bove may also be placed above the rendering node.

NOTES (continued)

4. The five nodes WINDOW, VIEWPORT, EYE, Field_Of_View, and MATRIX_4X4 should NOT, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWPORT, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiply-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR, SET PLOTTER) not mentioned above, are ignored by rendering operations when placed beneath a rendering node. Such nodes must therefore be placed above a rendering node if they are to have their customary effects on renderings. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPLAYed. If the object itself is DISPLAYed but its rendering node is not, no renderings can be created.

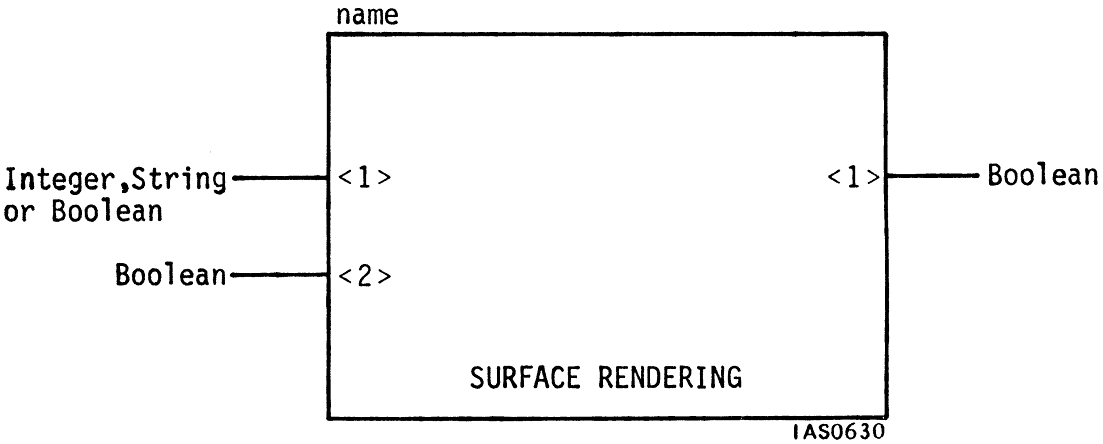
8. Any input to input<1> of a rendering node causes an output. Inputs sent to input<2> will not cause an output to be sent. If output<1> has not been connected, and an integer, string, or Boolean is sent to input<1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

DISPLAY TREE NODE CREATED

Rendering operation node.

(continued)

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

Input <1>
- 0: Toggles between the current rendering and the original object.
- 1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only).
- 2: Creates and displays a sectioned rendering.
- 3: Creates and displays a rendering using backface removal (solid only).
- 4: Creates and displays a rendering using hidden-line removal.
- 5: Generates a wash-shaded image on the raster display.
- 6: Generates a flat-shaded image on the raster display.
- 7: Generates a smooth-shaded image on the raster display.
- String: Causes the current rendering to be saved under the name given in the string.
- False: Sets the original view. The original descendant structure of the rendering operate node is displayed.
- True: Sets the rendered view. The rendered view of the original descendent structure of the rendering operate node.

Input <2>
- True: Declares the object to be a solid.
- False: Declares the object to be a surface.

Output <1>
- True: Rendering is displayed
- False: Rendering is not displayed

## FORMAT

```
name := TEXT SIZE x [APPLied to name1];
```

## DESCRIPTION

Creates a 2X2 uniform scale matrix which defines character size.

## PARAMETERS

x - The size of the characters.

name1 - Structure containing the characters.

## NOTES

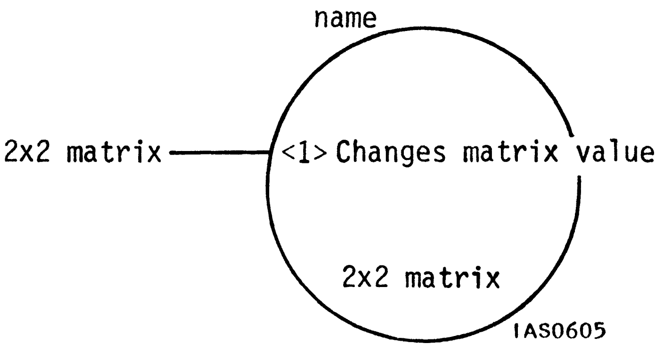
1. The text size (x) is a multiple or fraction of the default character size, i.e. 1.
2. A TEXT SIZE node in a display tree over-rides any previous character sizes that may have been created using the CHARacter SCAle or CHARachter SIZE commands. In other words, the TEXT SIZE scaling matrix is not concatenated into any other 2X2 matrix.
3. A TEXT SIZE node will also override CHARacter ROTate and MATRIX 2X2 nodes.

## DISPLAY TREE NODE CREATED

2x2 matrix operation node.

(continued)

## INPUTS FOR UPDATING NODE



## NOTE ON INPUT

Any 2x2 matrix is legal.

## ASSOCIATED FUNCTIONS

F:MATRIX2, F:CSCALE

## EXAMPLE

```
String := CHARacters 'This is only a test';
Scale := CHARacter SCAle 2 THEN String;
New_Scale := CHARacter SCAle 3 THEN Scale;
Change_Size := TEXT SIZE .5 THEN String;
```

{The Scale matrix creates characters twice the default size. The New_Scale matrix is concatenated with the Scale matrix to create characters six times the default size. The Change_Size matrix, however is not concatenated, and so returns the characters to one half of the default size.}

## FORMAT

name := TRANslate by tx,ty[,tz] [APPLied to name1];

## DESCRIPTION

Translates an object by applying a translation vector to it.

## PARAMETERS

tx,ty,tz – Distances to translate in each coordinate direction, in world coordinates.

name1 – Structure to be translated.

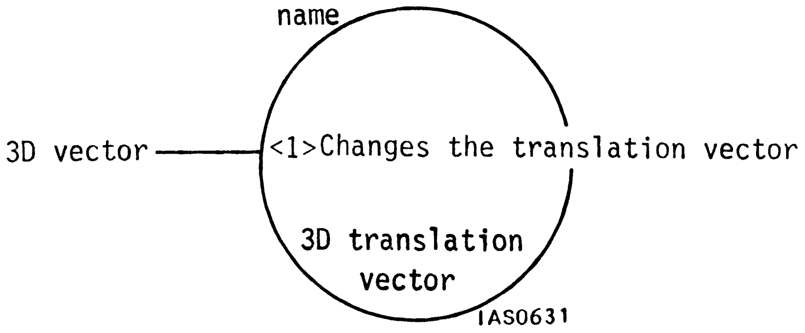
## DEFAULT

tz is 0 if not specified.

## DISPLAY TREE NODE CREATED

3D translation vector operation node.

## INPUTS FOR UPDATING NODE



(continued)

## ASSOCIATED FUNCTIONS

F:XVECTOR, F:YVECTOR, F:ZVECTOR

## EXAMPLE

```
A:= TRANslate by 5,7 THEN B;

B:= VECtor_list ... ;
```

## FORMAT

```
VARiable name1[,name2 ... namen];
```

## DESCRIPTION

Declares a storage place (or places) for any PS 300 Function data type. A value can be stored in variable **name1** either by SENDing (or STORing) a value to input <1> of **name1**, or by CONNECTing a Function Instance to input <1> of **name1**. The current value of variable **name1** can be obtained by using either the FETCH function or the SEND VALUE(variable) option of the SEND command, where variable in this case is **name1**.

## PARAMETERS

**name1**, **name2...** - Variable names.

## EXAMPLE

```
VARiable CURRENT_XY, X, Y, Z, SAVE;
```

FORMAT

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

DESCRIPTION

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

PARAMETERS

**name** - Any legal PS 300 name.

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

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

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

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

PARAMETERS (continued)

3. Connectivity:

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

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

C. DOTs - Each vector specifies a dot.

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

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

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

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

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

n - Estimated number of vectors.

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

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

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

(continued)

PARAMETERS (continued)

For ITEMized vector lists the syntax is:

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

or

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

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

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

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

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

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

DEFAULTS

If not specified, the options default to:

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

Non color-blended vectors default to:

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

If i is not specified, it defaults to 1.

Color-blended vectors default to:

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

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

(continued)

## NOTES

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

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

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

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

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

6. Options must be specified in the order given.

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

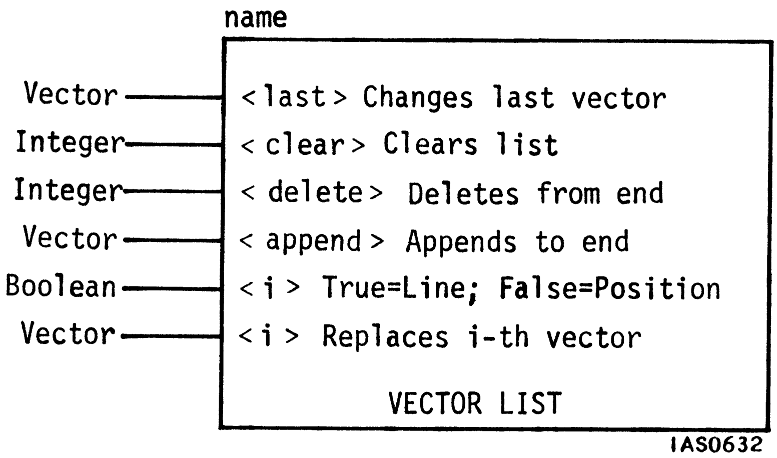
8. Block normalized vector lists generally take longer to process into the PS 300, but are processed faster for display once they are in the system.

## DISPLAY TREE NODE CREATED

Vector list data node.

(continued)

## INPUTS FOR UPDATING NODE



## NOTES ON INPUTS

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

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

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

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

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

## EXAMPLES

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

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

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

## FORMAT

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

## DESCRIPTION

Specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines.

## PARAMETERS

**hmin,hmax,vmin,vmax** - The x and y boundaries of the new viewport. Values must be within the -1 to 1 range relative to the current viewport, implying that each viewport may be no larger than its predecessor.

**imin,imax** - Specifies the minimum and maximum intensities for the viewport. **imin** is the intensity of lines at the back clipping plane; **imax** at the front clipping plane. Values must be within the 0 to 1 range relative to the current viewport, implying that each viewport may have no greater intensity range than its predecessor.

**name1** - Structure to which the viewport is applied.

## DEFAULT

The initial viewport is the full PS 300 screen with full intensity range (0 to 1):

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

(continued)

## NOTES

1. A new VIEWport is defined relative to the current viewport, whose boundaries are always taken to be -1 and 1 horizontally and vertically for the purposes of the command. (The "current" viewport is the one established by the most recent VIEWport command.)

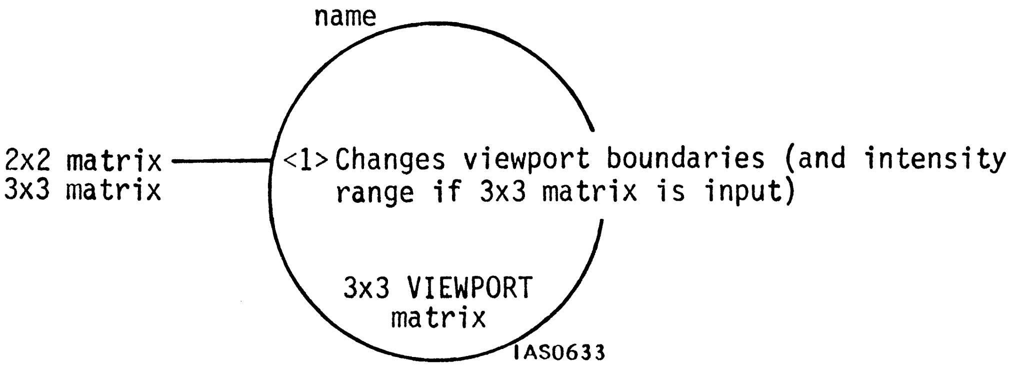
2. Viewports can be nested to any level.

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

## DISPLAY TREE NODE CREATED

3x3 viewport matrix operation node.

## INPUTS FOR UPDATING NODE


name
2x2 matrix
3x3 matrix
<1> Changes viewport boundaries (and intensity range if 3x3 matrix is input)
3x3 VIEWPORT matrix
IAS0633

## ASSOCIATED FUNCTIONS

F:MATRIX2, F:MATRIX3

(continued)

## NOTES ON INPUTS

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

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

## EXAMPLE

```
A:= VIEWport HORizontal = 0:1
         VERTical = 0:1
         INTENsity = .5:1 THEN B;
B:= ... ;
```

{If A is displayed, structure B will be displayed in the upper right quadrant of the screen with the intensity ranging from .5 to 1 instead of 0 to 1.}

## FORMAT

```
name := WINDOW X = xmin:xmax
               Y = ymin:ymax
               [FRONT boundary = zmin BACK boundary = zmax]
               [APPLied to name1];
```

## DESCRIPTION

Specifies a right rectangular prism enclosing a portion of the world coordinate system to be displayed in parallel projection (compare Field_Of_View).

## PARAMETERS

**xmin...zmax** - The window's boundaries along each axis (see Note 3.)

**name1** - Structure to which the window is applied.

## DEFAULT

WINDOW X=-1:1 Y=-1:1 FRONT=0 BACK=100000;

## NOTES

1. The windowing commands (WINDOW, Field_Of_View, and EYE) should always be the highest level element (the outermost transformation) in a display tree since these transformations override any previous transformations in the tree. Note that VIEWport is a mapping operation not a transformation of the data and thus is not affected by a windowing command.

2. These commands should also be followed by a LOOK command to fully specify the viewing transformation. (Refer to the LOOK command.)

(continued)

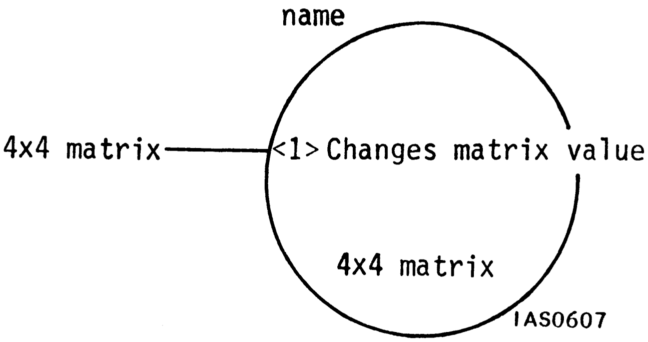
NOTES (continued)

3. The front and back boundaries should be specified relative to the AT point's position along the positive Z axis (0,0,D) (refer to the notes on the LOOK command). So, FRONT should equal (D minus **delta_min**) and BACK should equal (D plus **delta_max**), where **delta_min** and **delta_max** are the distances before and after the AT point that are to be included in the window, respectively. (See Note 3 of the LOOK command also.)

DISPLAY TREE NODE CREATED

4x4 matrix operation node.

INPUTS FOR UPDATING NODE



ASSOCIATED FUNCTIONS

F:WINDOW, F:FOV, F:MATRIX4

EXAMPLE

```
A:= BEGIN_Structure
      WINDOW X = -1:1 Y = -1:1
           FRONT boundary = 12
           BACK boundary = 14;
      LOOK AT 0,0,0 FROM 5,6.63,-10 THEN Sphere;
END_Structure;
```

{If Sphere is defined with a radius of 1 about the origin, A would be a view of the Sphere from 5,6.63,-10, fully depth-cued. Note that the FROM to AT distance in the LOOK command is 13.}

## FORMAT

```
name := WITH PATtern i [AROUND_corners] [MATCH/NOMATCH]
                LENgth r (VECtor_list);
```

## DESCRIPTION

Uses line patterns (dashes, center lines, etc.) in drawing a vector list. The line pattern is created over the length r, so lines will have the pattern repeated as many times as necessary to the end of the line.

## PARAMETERS

name - Any legal PS 300 name. A reference name for the patterned vector list.

i - A series of up to 32 integers between 0 and 128 indicating the relative lengths of alternating lines, spaces, lines, etc., in the pattern. The longer the series, the more complex the pattern of lines and spaces, which repeats every r units.

AROUND_corners - This indicates that patterning is to continue around each of the vectors in the vector list until the end of the list or a position vector is reached.

MATCH/NOMATCH - This indicates that the pattern length should be adjusted to make the pattern exactly match the end points of the vector or series of vectors being patterned. The default is MATCH.

r - The length over which i is defined and repeated.

VECtor_list -The standard VECtor_list command with all options available except DOTs.

## NOTES

1. The VECtor_list parameter n should be the estimate for the total number of vectors that will result from the command (not the number of vectors specified in the vector list).

NOTES (continued)

2. As r approaches 0, n approaches infinity.

3. If r is greater than a vector line segment, that segment will be drawn solid; no pattern will be used.

DISPLAY TREE NODE CREATED

Vector list data node.

INPUTS FOR UPDATING NODES

See VECtor_list command.

NOTES ON INPUTS

Remember that the vectors in the node are the patterned vectors, so it is non-trivial to update a vector.

EXAMPLES

```
WITH PATTERN 1 1 LENgth 1 VECtor_list N=2 0,0 3,0;

WITH PATTERN 1 1 LENgth 3 VECtor_list N=2 0,0 3,0;

WITH PATTERN 1 1 LENgth 4 VECtor_list N=2 0,0 3,0;

WITH PATTERN 1 1 1 1 LENgth 2 VECtor_list N=2 0,0 3,0;
```

{same as the first example}

```
WITH PATTERN 1 .25 .125 .25 .125 .25 1 LENgth 3
VECtor_list N=2 0,0 3,0;
```

## FORMAT

name := XFORM output_data_type APPLied to name1;

## DESCRIPTION

Allows transformed data to be saved either as a vector list or a 4x4 matrix at the point in the display tree where this XFORM data node is positioned.

## PARAMETERS

output_data_type - Specifies what type of transformed data is to be saved.

MATRIX -A single 4x4 matrix representing the concatenation of all transformation matrices currently in effect.

VECtor -A vector list specifying the transformed coordinates of the object (name1).

name1 - The object whose transformed data are to be saved.

## NOTE

This node indicates to the F:XFORMDATA function the point in the display tree where transformed data are requested.

## DISPLAY TREE NODE CREATED

XFORM operate node.

## ASSOCIATED FUNCTIONS

F:XFORMDATA, F:LIST, F:SYNC(2).

(continued)

EXAMPLE

```
XFORM := BEGIN_S  {Set up switch mechanism}
            X := SET CONDITIONAL_BIT 1 ON;
            IF CONDITIONAL_BIT 1 IS ON THEN VIEW;
            IF CONDITIONAL_BIT 1 IS OFF THEN TRAN;
          END_S;

TRAN := BEGIN_S  {To be:used while getting transformed data}
           MATRIX_4x4 1,0,0,0,0,1,0,0,0,1,0,0,0,1;
           INSTANCE OF OBJ;
          END_S

VIEW := BEGIN_S  {To be used while viewing and designing}
                   {Viewing commands: FIELD_OF_VIEW, WINDOW
                    EYE BACK, or 4x4_MATRIX}
          INSTANCE OF OBJ;
          END_S;

OBJ := BEGIN_S  {Setup transformed-data request}
       (Transformation commands:
       ROTATE, TRANSLATE, and/or SCALE)
       XFORM_REQUEST := XFORM VECTOR;
       INSTANCE OF DATA;
       END_S;


XFORMDATA := F:XFORMDATA;  {Build transformed-data network}
SYNC2 := F:SYNC(2);
LIST := F:LIST;
CONN SYNC2<1>:<1>XFORMDATA;
CONN XFORMDATA<1>:<1>LIST;
CONN LIST<1>:<1>HOST_MESSAGE;  {Send trans data to host}
CONN LIST<2>:<2>SYNC2;  {"Task completed" flag}
SEND <any message> TO <2>SYNC2;
SEND 'OBJ.XFORM_REQUEST' TO <2>XFORMDATA;
SEND 'XDATA' TO <3>XFORMDATA;
DISPLAY XFORM;
```

# APPENDIX A. PS 300 COMMANDS BY CATEGORY

## FUNCTION (Data Structuring)

(Function Instancing) name:=F:function_name
VARIABLE

## FUNCTION (Immediate-action)

CONNECT
DISCONNECT
SEND
SEND number*mode
SEND VL
SETUP CNESS
STORE

## GENERAL (Immediate-action)

### Command Control and Status:

BEGIN...END
COMMAND STATUS
OPTIMIZE MEMORY
OPTIMIZE STRUCTURE...END OPTIMIZE
REBOOT
RESERVE_WORKING_STORAGE
!RESET

## GENERAL (Immediate-action) (continued)

### Data Structuring and Display:

DELETE
DISPLAY
FORGET (structures)
FORGET (units)
REMOVE

### Initialization:

INITIALIZE

## HARDWARE ATTRIBUTES (Immediate-action)

ALLOCATE PLOTTER
DEALLOCATE PLOTTER

## MODELING (Data Structuring)

### Character Transformations:

CHARACTER SCALE
CHARACTER ROTATE
MATRIX_2x2
TEXT SIZE

### Picking Attributes:

SET PICKING
SET PICKING IDENTIFIER
SET PICKING LOCATION

### Primitives:

BEGIN_FONT...END_FONT
BSPLINE
CHARACTERS
COPY
ERASE PATTERN FROM
LABELS
PATTERN
PATTERN WITH

## MODELING (Data Structuring) (continued)

**Primitives:** (continued)

POLYGON
POLYNOMIAL
RATIONAL BSPLINE
RATIONAL POLYNOMIAL
VECTOR_LIST
WITH PATTERN

**Transformed Data Attributes:**

XFORM

**Transformations:**

MATRIX_3x3
MATRIX_4x3
MATRIX_4x4
ROTATE
SCALE
TRANSLATE

## RENDERING (Data Structuring)

ATTRIBUTES
ILLUMINATION
SOLID_RENDERING
SURFACE_RENDERING

## STRUCTURE (Data Structuring)

**Attributes:**

DECREMENT LEVEL_OF_DETAIL
INCREMENT LEVEL_OF_DETAIL
SET CONDITIONAL_BIT
SET LEVEL_OF_DETAIL
SET RATE
SET RATE EXTERNAL

## STRUCTURE (Data Structuring) (continued)

### Conditional Referencing:

IF CONDITIONAL_BIT
IF LEVEL_OF_DETAIL
IF PHASE

### Explicit Referencing:

APPLIED TO/THEN
INSTANCE OF
NAME:=

### Implicit Referencing:

BEGIN_STRUCTURE...END_STRUCTURE

## STRUCTURE (Immediate-action)

### Modifying:

FOLLOW WITH
INCLUDE
PREFIX WITH
REMOVE FOLLOWER
REMOVE FROM
REMOVE PREFIX

## VIEWING (Data Structuring)

### Appearance Attributes:

CHARACTER FONT
SET CHARACTERS
SET COLOR
SET COLOR BLENDING
SET CONTRAST
SET CSM
SET DEPTH_CLIPPING
SET DISPLAYS
SET PLOTTER
STANDARD FONT

## VIEWING (Data Structuring) (continued)

**Viewport Specification:**

SET INTENSITY
VIEWPORT

**Windowing Transformations:**

EYE
FIELD_OF_VIEW
LOOK
WINDOW

## PS 340-SPECIFIC COMMANDS

ATTRIBUTES
ILLUMINATION
POLYGON
RESERVE_WORKING_STORAGE
SECTIONING_PLANE
SOLID_RENDERING
SURFACE_RENDERING

# APPENDIX B. PS 300 COMMAND SYNTAX

## ALLOCATE PLOTTER

```
   ALLOCATE PLOTTER device_number;
```

## APPLIED TO/THEN

```
   name := operation_command [APPLied to name1];
   name := operation_command [THEN name1];
```

## ATTRIBUTES

```
   name := ATTRIBUTES attributes [AND attributes];
```

## BEGIN...END

```
   BEGIN
     command;
     command;
     .
     .
     .
     command;
   END;
```

### BEGIN_FONT...END_FONT

```
   name := BEGIN_Font
              [C[0]: N=n {itemized 2D vectors};]
                         .
                         .
                         .
              [C[i]: N=n {itemized 2D vectors};]
                         .
                         .
                         .
              [C[127]: N=n {itemized 2D vectors};]
           END_Font;
```

### BEGIN_S ... END_S

```
   name :=  BEGIN_Structure
   [name1:=]     nameable_command;
                     .
                     .
                     .
   [namen:=]     nameable_command;
         END_Structure;
```

### BSPLINE

```
   name := BSpline ORDER=k
              [OPEN/CLOSED] [NONPERIodic/PERIodic] [N=n]
              [VERTICES =] x1,y1,[z1]
                       x2,y2,[z2]
                              .   .   .
                              .   .   .
                              .   .   .
                       xn,yn,[zn]
              [KNOTS = t1,t2,. . .,tj]
              CHORDS = q;
```

### CHARACTER FONT

```
   name := character FONT font_name [APPLied to name1];
```

### CHARACTER ROTATE

```
   name := CHARacter ROTate angle [APPLied to name1];
```

### CHARACTER SCALE

```
name := CHARacter SCAle s [APPLied to name1];
name := CHARacter SCAle sx,sy [APPLied to name1];
```

### CHARACTERS

```
name := CHARacters [x,y[,z]][STEP dx,dy] 'string';
```

### COMMAND STATUS

```
COMmand STATus;
```

### CONNECT

```
CONNect name1<i>:<j>name2;
```

### COPY

```
name := COPY name1 [START=] i [,] [COUNT=] n;
```

### DEALLOCATE PLOTTER

```
DEALLOCATE PLOTTTER device_number;
```

### DECREMENT LEVEL_OF_DETAIL

```
name:= DECrement LEVel_of_detail[APPLied to name1];
```

### DELETE

```
DELete name[,name1 ... namen];
DELete any_string*;
```

### DISCONNECT

```
DISCONNect name1[<i>]:option;
```

### DISPLAY

```
DISPlay name;
```

### ERASE PATTERN FROM

```
ERASE PATTERN FROM name;
```

### EYE

```
name := EYE BACK z [option1][option2] from SCREEN area w WIDE
           [FRONT boundary = zmin BACK boundary = zmax]
           [APPLied to name1];
```

### FIELD_OF_VIEW

```
name := Field_Of_View angle
           [FRONT boundary = zmin BACK boundary = zmax]
           [APPLied to name1];
```

### FOLLOW WITH

```
FOLLOW name WITH option;
```

### FORGET (structures)

```
FORget name;
```

### FORGET (units)

```
FORget (unit_name);
```

### (Function Instancing)

```
name := F:function_name;
```

### IF CONDITIONAL_BIT

```
name := IF conditional_BIT n is state [THEN name1];
```

### IF LEVEL_OF_DETAIL

```
name := IF LEVel_of_detail relationship n [THEN name1];
```

### IF PHASE

```
name := IF PHASE is state THEN [name1];
```

### ILLUMINATION

```
name := ILLUMINATION x,y,z [COLOR h [,s [,i]]] [AMBIENT a];
```

### INCLUDE

```
INCLude name1 IN name2;
```

### INCREMENT LEVEL_OF_DETAIL

```
name:= INCRement LEVel_of_detail[APPLied to name1];
```

### INITIALIZE

```
INITialize [option];
```

### INSTANCE OF

```
name := INSTance of name1[,name2 ... namen];
```

### LABELS

```
name := LABELS x,y,[,z] 'string'[xi,yi[zi] 'string'... ];
```

### LOOK

```
name := LOOK AT ax,ay,az FROM fx,fy,fz
          [UP ux,uy,uz] [APPLied to name1];

name := LOOK FROM fx,fy,fz AT ax,ay,az
          [UP ux,uy,uz] [APPLied to name1];
```

### MATRIX_2x2

```
name := Matrix_2x2   m11,m12
                     m21,m22 [APPLied to name1];
```

### MATRIX_3x3

```
name := Matrix_3x3   m11,m12,m13
                     m21,m22,m23
                     m31,m32,m33 [APPLied to name1];
```

### MATRIX_4x3

```
name := Matrix_4x3   m11,m12,m13
                     m21,m22,m23
                     m31,m32,m33
                     m41,m42,m43 [APPLied to name1];
```

### MATRIX_4x4

```
name := Matrix_4x4   m11,m12,m13,m14
                     m21,m22,m23,m24
                     m31,m32,m33,m34
                     m41,m42,m43,m44 [APPLied to name1];
```

### (Naming of Display Data Structures)

```
name:= display_data_structure_command;
```

### OPTIMIZE MEMORY

```
OPTIMIZE MEMORY;
```

### OPTIMIZE STRUCTURE; .... END OPTIMIZE;

```
    OPTIMIZE STRUCTURE;
              command;
              command;
                .
                .
    END OPTIMIZE;
```

### PATTERN

```
name := PATtern i [AROUND_corners][MATCH/NOMATCH] LENgth r;
```

### PATTERN WITH

```
PATTERN name1 WITH pattern;
```

### POLYGON

```
name := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [COPLANAR]
        POLYGon vertex ... vertex;
```

### POLYNOMIAL

```
name:= POLYnomial[ORDER=i]
        [COEFFICIENTS=] xi,    yi,    zi
                        xi-1,  yi-1,  zi-1
                         .      .      .
                         .      .      .
                         .      .      .
                        x0,    y0,    z0
        CHORDS= q;
```

### PREFIX WITH

```
PREFIX name WITH operation_command;
```

### RATIONAL BSPLINE

```
name := RATIonal BSpline ORDER=k
        [OPEN/CLOSED] [NONPERIodic/PERIodic] [N=n]
        [VERTICES =]   x1,y1,[z1],w
                       x2,y2,[z2],w2
                        .   .   .    .
                        .   .   .    .
                        .   .   .    .
                       xn,yn,[zn],wn
        [KNOTS = t1,t2,...,tj]
        CHORDS = q;
```

**RATIONAL POLYNOMIAL**

```
name:= RATional POLYnomial[ORDER=i]
       [COEFFICIENTS=]  xi,     yi,     zi,     wi
                        xi-1,   yi-1,   zi-1,   wi-1
                         .       .       .       .
                         .       .       .       .
                         .       .       .       .
                        x0,     y0,     z0,     w0
        CHORDS= q;
```

**REBOOT**

```
name := REBOOT password;
```

**REMOVE**

```
REMove name;
```

**REMOVE FOLLOWER**

```
REMove FOLLOWER of name;
```

**REMOVE FROM**

```
REMove name1 FROM name2;
```

**REMOVE PREFIX**

```
REMove PREfix of name;
```

**RESERVE_WORKING_STORAGE**

```
RESERVE_WORKING_STORAGE size;
```

**ROTATE**

```
name := ROTate in [axis] angle [APPLied to name1];
```

### SCALE

name := SCALE by s [APPLied to name1];
name := SCALE by sx,sy[,sz] [APPLied to name1];

### SEND

SEND option TO <n>name1;

### SEND number*mode

SEND number*mode TO <n>name1;

### SEND VL

SEND VL (name1) TO <i>name 2;

### SET CHARACTERS

name := SET CHARacters orientation [APPLied to name1];

### SET COLOR

name := SET COLOR hue,sat [APPLied to name1];

### SET COLOR BLENDING

name := SET COLOR BLENDing sat [APPLied to name1];

### SET CONDITIONAL_BIT

name := SET conditional_BIT n switch [APPLied to name1];

### SET CONTRAST

name := SET CONTrast to c [APPLied to name1];

### SET CSM

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

### SET DEPTH_CLIPPING

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

### SET DISPLAYS

```
name := SET DISPlays ALL switch [APPLied to name1];
name := SET DISPlay n[,m...] switch [APPLied to name1];
```

### SET INTENSITY

```
name := SET INTENsity switch imin:imax [APPLied to name1];
```

### SET LEVEL_OF_DETAIL

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

### SET PICKING

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

### SET PICKING IDENTIFIER

```
name := SET PICKing IDentifier = id_name
          [APPLied to name1];
```

### SET PICKING LOCATION

```
name := SET PICKing LOCation = x,y size_x,size_y;
```

### SET PLOTTER

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

## SET RATE

```
name := SET RATE phase_on phase_off [initial_state] [delay]
          [APPLied to name1];
```

## SET RATE EXTERNAL

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

## SETUP CNESS

```
SETUP CNESS queue_type <i>name;
```

## SOLID_RENDERING

```
name := SOLID_rendering APPLied to name1;
```

## STANDARD FONT

```
name := STANdard FONT [APPLied to name1];
```

## STORE

```
STORE option IN name1;
```

## SURFACE_RENDERING

```
name := SURFACE_rendering APPLied to name1;
```

## TEXT SIZE

```
name := TEXT SIZE x [APPLIED to name1];
```

## TRANSLATE

```
name := TRANslate by tx,ty[,tz] [APPLied to name1];
```

### VARIABLE

```
VARiable name1[,name2 ... namen];
```

### VECTOR_LIST

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

### VIEWPORT

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

### WINDOW

```
name := WINDOW X = xmin:xmax Y = ymin:ymax
           [FRONT boundary = zmin BACK boundary = zmax]
           [APPLied to name1];
```

### WITH PATTERN

```
name := WITH PATtern i [AROUND_corners][MATCH/NOMATCH]
           LENgth r VECtor_list;
```

### XFORM

```
name := XFORM output_data_type [APPLied to name1];
```

### !RESET

```
!RESET;
```

# APPENDIX C. PS 300 ASCII COMMANDS AND CORRESPONDING GSRs

The following list from left to right gives an alphabetical listing of the PS 300 ASCII Command Name, the Pascal Application Procedure Name, and the FORTRAN Subroutine Call.

| ASCII COMMAND NAME | Pascal PROCEDURE | FORTRAN SUBROUTINE |
|---|---|---|
| ALLOCATE PLOTTER | PALLPLOT | PALLPL |
| ATTRIBUTES | PATTRIB<br>PATTRIB2 | PATTR<br>PATTR2 |
| BEGIN | PBEGIN | PBEG |
| BEGIN_STRUCTURE | PBEGINS | PBEGS |
| BSPLINE | PBSPL | PBSPL |
| CHARACTER FONT | PFONT | PFONT |
| CHARACTER ROTATE | PCHARROT | PCHROT |
| CHARACTERS [STEP] | PCHARS | PCHS |
| CHARACTER SCALE | PCHARSCA | PSCHSC |
| CONNECT | PCONNECT | PCONN |
| COPY | PCOPYVEC | PCOPYV |
| DEALLOCATE PLOTTER | PDALLPLT | PDALLP |

| ASCII COMMAND NAME | Pascal PROCEDURE | FORTRAN SUBROUTINE |
|---|---|---|
| DECREMENT LEVEL_OF_DETAIL | PDECLOD | PDELOD |
| DELETE | PDELETE | PDELET |
| DELETE NAME* | PDELWILD | PDELW |
| DISCONNECT ALL | PDISCALL | PDIALL |
| DISCONNECT | PDISC | PDI |
| DISCONNECT OUTPUT | PDISCOUT | PDIOUT |
| DISPLAY | PDISPLAY | PDISP |
| END | PEND | PEND |
| END_STRUCTURE | PENDS | PENDS |
| END OPTIMIZE | PENDOPT | PENDOP |
| ERASE PATTERN | PERAPATT | PERAPA |
| EYE | PEYEBACK | PEYEBK |
| F:FUNCTION NAME | PFNINST | PFN |
| FOLLOW WITH | PFOLL | PFOLL |
| FORGET | PFORGET | PFORG |
| FIELD_OF_VIEW | PFOV | PFOV |
| IF CONDITIONAL_BIT | PIFBIT | PIFBIT |
| IF LEVEL_OF_DETAIL | PIFLEVEL | PIFLEV |
| IF PHASE | PIFPHASE | PIFPHA |
| ILLUMINATION | PILLUMIN | PILLUM |
| INCLUDE | PINCL | PINCL |
| INCREMENT LEVEL_OF_DETAIL | PINCLOD | PINLOD |
| INITIALIZE | PINIT | PINIT |

| ASCII COMMAND NAME | Pascal PROCEDURE | FORTRAN SUBROUTINE |
|---|---|---|
| INITIALIZE CONNECTIONS | PINITC | PINITC |
| INITIALIZE DISPLAYS | PINITD | PINITD |
| INITIALIZE NAMES | PINITN | PINITN |
| INSTANCE OF | PINST | PINST |
| LABELS | PLABBEGN<br>PLABADD<br>PLABEND | PLABEG<br>PLAADD<br>PLAEND |
| LOOK AT FROM | PLOOKAT | PLOOKA |
| MATRIX_2X2 | PMAT2X2 | PMAT22 |
| MATRIX_3X3 | PMAT3X3 | PMAT33 |
| MATRIX_4X3 | PMAT4X3 | PMAT43 |
| MATRIX_4X4 | PMAT4X4 | PMAT44 |
| NAME:= NIL | PNAMENIL | PNIL |
| NAME:= PATTERN | PDEFPATT | PDEFPA |
| OPTIMIZE STRUCTURE | POPTSTRU | POPT |
| PATTERN WITH | PPATWITH | PPATWI |
| POLYGON (ATTRIBUTES) | PPLYGATR | PPLYGA |
| POLYGON (BEGIN) | PPLYGBEG | PPLYBG |
| POLYGON (END) | PPLYGEND | PPLYGE |
| POLYGON (LIST) | PPLYGLIS | PPLYGL |
| POLYGON (OUTLINE) | PPLYGOTL | PPLYGO |
| POLYNOMIAL | PPOLY | PPOLY |
| PREFIX NAME WITH | PPREF | PPREF |
| RATIONAL BSPLINE | PRBSPL | PRBSPL |

| ASCII COMMAND NAME | Pascal PROCEDURE | FORTRAN SUBROUTINE |
|---|---|---|
| REMOVE NAME | PREM | PREM |
| REMOVE FOLLOWER OF NAME | PREMFOLL | PREMFO |
| REMOVE FROM | PREMFROM | PREMFR |
| REMOVE PREFIX | PREMPREF | PREMPR |
| ROTATE IN X | PROTX | PROTX |
| ROTATE IN Y | PROTY | PROTY |
| ROTATE IN Z | PROTZ | PROTZ |
| RATIONAL POLYNOMIAL | PRPOLY | PRPOLY |
| RESERVE_WORKING_STORAGE | PRSVSTOR | PRSVST |
| SCALE | PSCALEBY | PSCALE |
| SECTIONING_PLANE | PSECPLAN | PSECPL |
| SEND BOOLEAN TO | PSNDBOOL | PSNBOO |
| SEND FIX TO | PSNDFIX | PSNFIX |
| SEND 2x2 MATRIX TO | PSNDM2D | PSNM2D |
| SEND 3x3 MATRIX TO | PSNDM3D | PSNM3D |
| SEND 4x4 MATRIX TO | PSNDM4D | PSNM4D |
| SEND NUMBER*MODE TO | PSNDPL | PSNPL |
| SEND REAL NUMBER TO | PSNDREAL | PSNREA |
| SEND STRING TO | PSNDSTR | PSNST |
| SEND 2D VECTOR TO | PSNDV2D | PSNV2D |
| SEND 3D VECTOR TO | PSNDV3D | PSNV3D |
| SEND 4D VECTOR TO | PSNDV4D | PSNV4D |
| SEND VALUE TO | PSNDVAL | PSNVAL |

| ASCII COMMAND NAME | Pascal PROCEDURE | FORTRAN SUBROUTINE |
|---|---|---|
| SEND VECTOR LIST | PSNDVL | PSNVL |
| SET CONDITIONAL_BIT | PSETBIT | PSEBIT |
| SET CHARACTERS SCREEN_ORIENTED | PSETCHRS | PSECHS |
| SET CHARACTERS SCREEN_ORIENTED/FIXED | PSETCHRF | PSECHF |
| SET CHARACTERS WORLD_ORIENTED | PSETCHRW | PSECHW |
| SET COLOR | PSETCOLR | PSECOL |
| SET COLOR BLENDING | PSETBLND | PSETCB |
| SET CONTRAST | PSETCONT | PSECON |
| SET CSM | PSETCSM | PSECSM |
| SET DISPLAYS ALL | PSETDALL | PSEDAL |
| SET DEPTH_CLIPPING | PSETDCL | PSEDCL |
| SET DISPLAY | PSETDONF | PSEDOF |
| SET INTENSITY | PSETINT | PSEINT |
| SET LEVEL_OF_DETAIL | PSETLOD | PSELOD |
| SET PICKING INDENTIFIER | PSETPID | PSEPID |
| SET PICKING LOCATION | PSETPLOC | PSEPLO |
| SET PICKING OFF | PSETPONF | PSEPOF |
| SET RATE | PSETR | PSER |
| SET RATE EXTERNAL | PSETREXT | PSEREX |
| SETUP CNESS | PSETCNES | PSECNS |
| SOLID_RENDERING | PSOLREND | PSOLRE |
| SURFACE_RENDERING | PSURREND | PSURRE |

| ASCII COMMAND NAME | Pascal PROCEDURE | FORTRAN SUBROUTINE |
|---|---|---|
| STANDARD FONT | PSTDFONT | PSTDFO |
| TRANSLATE | PTRANSBY | PTRANS |
| VARIABLE NAME | PVAR | PVAR |
| VECTOR_LIST | PVECBEGN<br>PVECLIST<br>PVECEND | PVCBEG<br>PVCLIS<br>PVCEND |
| VIEWPORT | PVIEWP | PVIEWP |
| WINDOW | PWINDOW | PWINDO |
| XFORM MATRIX | PXFMATRX | PXFMAT |
| XFORM VECTOR | PXFVECTR | PXFVEC |

# ASCII Character Code Set

| Decimal Value | ASCII Character | Decimal Value | ASCII Character | Decimal Value | ASCII Character |
|---|---|---|---|---|---|
| 0 | NUL | 44 | , | 88 | X |
| 1 | SOH | 45 | - | 89 | Y |
| 2 | STX | 46 | . | 90 | Z |
| 3 | ETX | 47 | / | 91 | [ |
| 4 | EOT | 48 | 0 | 92 | \ |
| 5 | ENQ | 49 | 1 | 93 | ] |
| 6 | ACK | 50 | 2 | 94 | ↑ or ^ |
| 7 | BEL | 51 | 3 | 95 | ← or _ |
| 8 | BS | 52 | 4 | 96 | ` |
| 9 | HT | 53 | 5 | 97 | a |
| 10 | LF | 54 | 6 | 98 | b |
| 11 | VT | 55 | 7 | 99 | c |
| 12 | FF | 56 | 8 | 100 | d |
| 13 | CR | 57 | 9 | 101 | e |
| 14 | SO | 58 | : | 102 | f |
| 15 | SI | 59 | ; | 103 | g |
| 16 | DLE | 60 | < | 104 | h |
| 17 | DC1 | 61 | = | 105 | i |
| 18 | DC2 | 62 | > | 106 | j |
| 19 | DC3 | 63 | ? | 107 | k |
| 20 | DC4 | 64 | @ | 108 | l |
| 21 | NAK | 65 | A | 109 | m |
| 22 | SYN | 66 | B | 110 | n |
| 23 | ETB | 67 | C | 111 | o |
| 24 | CAN | 68 | D | 112 | p |
| 25 | EM | 69 | E | 113 | q |
| 26 | SUB | 70 | F | 114 | r |
| 27 | ESC or ALT | 71 | G | 115 | s |
| 28 | FS | 72 | H | 116 | t |
| 29 | GS | 73 | I | 117 | u |
| 30 | RS | 74 | J | 118 | v |
| 31 | VS | 75 | K | 119 | w |
| 32 | SP | 76 | L | 120 | x |
| 33 | ! | 77 | M | 121 | y |
| 34 | " | 78 | N | 122 | z |
| 35 | # | 79 | O | 123 | { |
| 36 | $ | 80 | P | 124 | \| |
| 37 | % | 81 | Q | 125 | } |
| 38 | & | 82 | R | 126 | ~ Tilde |
| 39 | ' | 83 | S | 127 | Rubout or DEL |
| 40 | ( | 84 | T | | |
| 41 | ) | 85 | U | | |
| 42 | * | 86 | V | | |
| 43 | + | 87 | W | | |

# PS 300 FUNCTION SUMMARY

Many concepts in this document are proprietary to Evans & Sutherland, and are protected as trade secrets or covered by U.S. and foreign patents or patents pending.

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

# PREFACE

This manual is a PS 300 Function reference guide for users who are already familiar with the basic operation of the PS 300.

There are three types of PS 300 functions: Intrinsic Functions, Initial Function Instances, and User-Written Functions. This document is a reference for the first two types only. User-Written Functions are documented in Volume 4 of the PS 300 Documentation Set.

Also included in this reference are the Initial Structures CURSOR and PICK_LOCATION. These establish the shape of the cursor as an 'X' and the pick-sensitive location as the center of the cursor.

## CONTENTS

### Intrinsic Functions

## Intrinsic Functions (continued)

## Intrinsic Functions (continued)

## Intrinsic Functions (continued)

## Intrinsic Functions (continued)

## Intrinsic Functions (continued)

## Initial Function Instances

Initial Function Instances (continued)

Initial Structures

### Table 1. Key to Abbreviations for Valid Data Types

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

## Conjunctive/Disjunctive Sets

Inputs and outputs to a function are either disjunctive or conjunctive. The following notation is used in the Function Summary to indicate disjunctive or conjunctive inputs and outputs.

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

## Intrinsic Functions

Intrinsic Functions are the master set of function "templates" which are available for the user to instance and use in building Function Networks. These functions are of the form

```
F:Identifier
```

where "identifier" is the name of the function, e.g. ROUTE, MUL, CONCATENATE. Using the ***NAME := F:Identifier***; command, the user can create uniquely named instances of Intrinsic Functions. For example,

```
Adder := F:ADD;
```

creates a function called Adder which is a uniquely named instanre of the F:ADD Intrinsic Function. Input queues and outputs of user-instanced functions are connected to create Function Networks for handling data input from the Interactive Devices, from the host computer, or from other functions. For example,

```
CONNECT Adder<1>:<1>Multiply;
```

connects output 1 of the function instance Adder to input queue 1 of the function instance Multiply.

## Initial Function Instances

Whenever the PS 300 is initialized, certain Initial Function Instances are loaded into memory with the Graphics Firmware. Initial Function Instances are of the form:

```
Function_instance_name
```

Unlike Intrinsic Functions, they are not preceded by "F:", e.g. TABLETIN, OFFBUTTONLIGHTS. They provide access to host communication and to PS 300 interactive devices, as well as allowing for the display of messages on keyboard and control dial LEDs. Initial Function Instances are not used as templates to create uniquely-named function instances. Instead, they are used in Function Networks by their own system-assigned name. They cannot be renamed by the user. For example,

```
SEND 'EXIT     ' TO <1>FLABEL12;
```

sends the string EXIT to the LED for Function Key 12.

## Reference Documentation

Each Intrinsic Function and Initial Function Instance in the PS 300 firmware is concisely documented in this Function Summary. Intrinsic Functions are listed first, then Initial Function Instances. Functions are ordered alphabetically. The function name appears in the upper right corner of each page. The type of function (e.g. Intrinsic) and its category (e.g. Data Conversion) are shown in the upper left corner. Appendix A lists functions by category. Appendix B lists display tree nodes that can accept input data from a function, the commands that create these nodes, and the data types which the nodes accept. Since some functions use the ASCII decimal equivalent of characters, an ASCII chart with decimal codes is included after the appendices.

The following information, where relevant, is given for each function:

Name
Type
Category
Purpose
Description of inputs and outputs
Defaults
Notes
Associated functions
Examples

## Function Representation

Functions are represented as 'black boxes' with numbered input queues and outputs enclosed in angle brackets. Valid data types are shown in abbreviated form at each input and output. A "C" in the function name usually indicates that one or more input queues contain a constant value. A constant input is shown by the letter "C" following the input number in angle brackets.

The following is a key to the abbreviations used.

```
                          ______________________
                         |                      |
                         |     F:ACCUMULATE     |
                         |                      |
R, 2D, 3D, 4D, B ---->   |<1>                <1>|-----> R, 2D, 3D, 4D
                         |                      |
R, 2D, 3D, 4D ------->   |<2> C                 |
                         |                      |
R ------------------->   |<3> C                 |
                         |                      |
R, 2D, 3D, 4D ------->   |<4> C                 |
                         |                      |
R, 2D, 3D, 4D ------->   |<5> C                 |
                         |                      |
R, 2D, 3D, 4D ------->   |<6> C                 |
                         |                      |
                         |         D D          |
                         |______________________|
```

## PURPOSE

Accumulates a series of input values and sends the sum at specified intervals.

## DESCRIPTION

INPUT
<1> - value to be accumulated
<2> - initial value (constant)
<3> - output interval (constant)
<4> - scale factor (constant)
<5> - upper limit on sum (constant)
<6> - lower limit on sum (constant)

OUTPUT
<1> - sum

(continued)

## NOTES

1. The input values may be scaled, and the output values may be limited to a specified range as in F:LIMIT. Note that this combination of operations is especially useful for handling input from the control dials.

2. An initial value must be sent to input <2>; subsequent values are sent to input <1>. All values at input <1> are scaled by input <4> before adding.

3. The sum is output whenever it differs from the previous F:ACCUMULATE output (or zero if there was no previous output) by more than the value at input <3>. (If vectors are being accumulated, this difference and the value at input <3> are taken to be vector lengths, and, therefore, real numbers. Vector lengths are considered to be $n(x,y) = |x| + |y|$, not $n(x,y) = x^2 + y^2$.)

4. Inputs <5> and <6> specify limits (upper and lower, respectively) to be applied to the accumulated sum. A sum falling outside the range is adjusted to the nearer limit, and any further accumulations operate on the limited sum.

5. Inputs <1> and <2> must be of the same data type. To change the data type of the sum to be accumulated, send a new initial value of the appropriate type to <2>. Note that the data type of the accumulated sum may not be changed simply by starting to send different data types to <1>--these will only generate an "Incompatible inputs" error message.

6. If input <2> is real, then inputs <4>, <5>, and <6> must be real. On the other hand, if input <2> is a vector, then each of inputs <4>, <5>, and <6> may be either a vector of the same dimension as <2> or a real number.

7. If vectors are being accumulated, but the scale factor at <4> is real, then each coordinate of each vector accumulated at <1> is multiplied by the real scale factor before the vector is added in. If the scale factor at <4> is a vector, each of its coordinates is multiplied by the corresponding coordinate of the accumulated vector.

8. If vectors are being accumulated, but both the upper sum limit at <5> and the lower sum limit at <6> are real, then these real numbers are the limits for each coordinate of the sum. If <5> and <6> are vectors, each of its coordinates is applied as a limit to the corresponding coordinate of the sum.

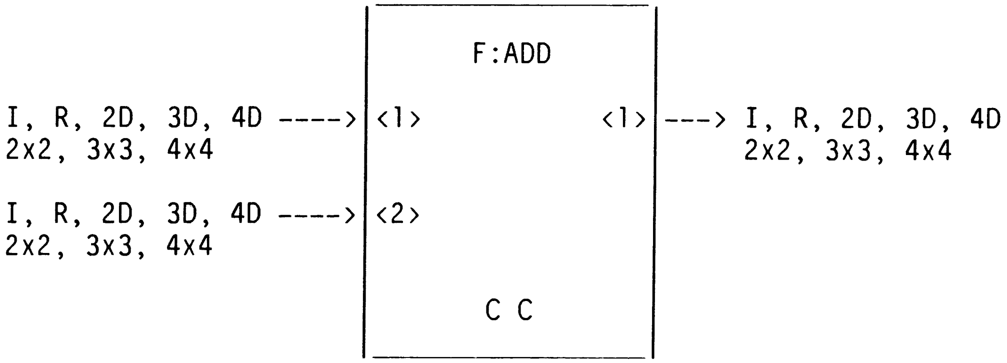
NOTES (continued)

9. If input <1> is Boolean (regardless of value), the current sum is immediately sent to output <1>.

10. Vector types may not be mixed in an **F:ACCUMULATE** operation; all vectors must be either 2D, 3D, or 4D.

EXAMPLE

Refer to Application Note 10 in the PS 300 Application Notes.



```
                                 F:ADD
I, R, 2D, 3D, 4D ----> <1>                 <1> ---> I, R, 2D, 3D, 4D
2x2, 3x3, 4x4                                       2x2, 3x3, 4x4

I, R, 2D, 3D, 4D ----> <2>
2x2, 3x3, 4x4

                                  C C
```

## PURPOSE

Accepts two inputs and produces an output that is the sum of those inputs.

## DESCRIPTION

INPUT
<1> - input value
<2> - input value

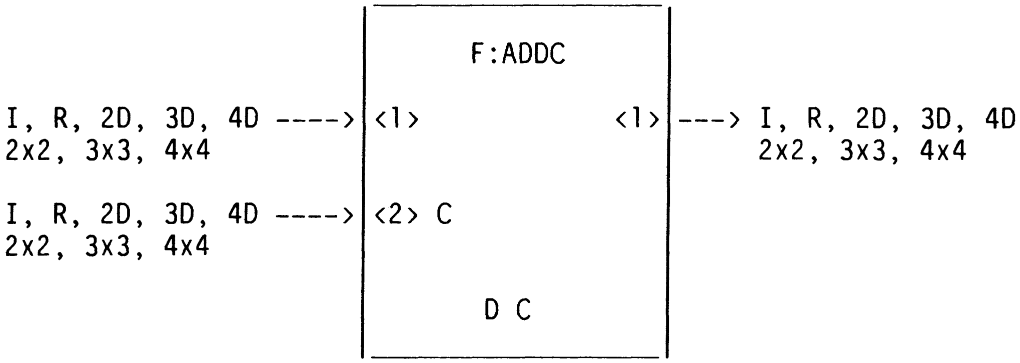
OUTPUT
<1> - sum

## NOTES

The two input values must be of the same data type (except a combination of real and integer is allowed); the output data type depends on the input data type(s). If an integer is added to a real number the output is a real number.

## ASSOCIATED FUNCTIONS

F:ADDC


F:ADDC
I, R, 2D, 3D, 4D ----> <1>
2x2, 3x3, 4x4
I, R, 2D, 3D, 4D ----> <2> C
2x2, 3x3, 4x4
<1> ---> I, R, 2D, 3D, 4D
2x2, 3x3, 4x4
D C

## PURPOSE

Accepts two inputs and produces an output that is the sum of those inputs. Input <2> is a constant.

## DESCRIPTION

INPUT
<1> – input value
<2> – input value (constant)

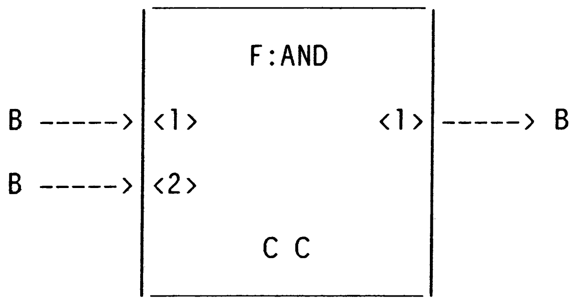
OUTPUT
<1> – sum

## NOTES

The two input values must be of the same data type (except a combination of real and integer is allowed); the output data type depends on the input data type(s). If an integer is added to a real number the output is a real number.

## ASSOCIATED FUNCTIONS

F:ADD



## PURPOSE

Accepts two Booleans as input and produces a Boolean output that is the logical AND of the two inputs.

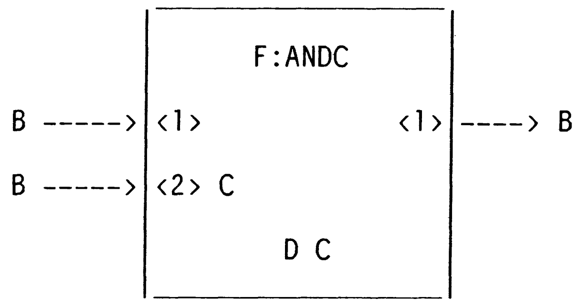
## DESCRIPTION

INPUT
<1> - Boolean input
<2> - Boolean input

OUTPUT
<1> - logical AND of the two inputs

## ASSOCIATED FUNCTIONS

**F:ANDC**



## PURPOSE

Accepts two Booleans as input and produces a Boolean output that is the logical AND of the two inputs. Input <2> is a constant.

## DESCRIPTION

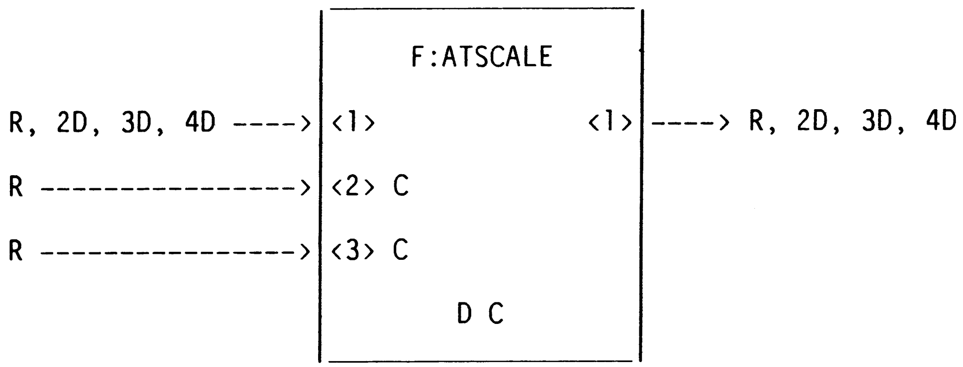
INPUT
<1> - Boolean input
<2> - Boolean input (constant)

OUTPUT
<1> - logical AND of the two inputs

## ASSOCIATED FUNCTIONS

F:AND

# F:ATSCALE



## PURPOSE

Like F:ACCUMULATE, F:ATSCALE accumulates the sum of a series of real numbers or vectors. Unlike F:ACCUMULATE, its sum is cleared after output.

## DESCRIPTION

INPUT
<1> – value to be accumulated
<2> – scale factor (constant)
<3> – delta (constant)

OUTPUT
<1> – accumulated sum

## DEFAULT

Input <2> = 1.0, Input <3> = 0.0

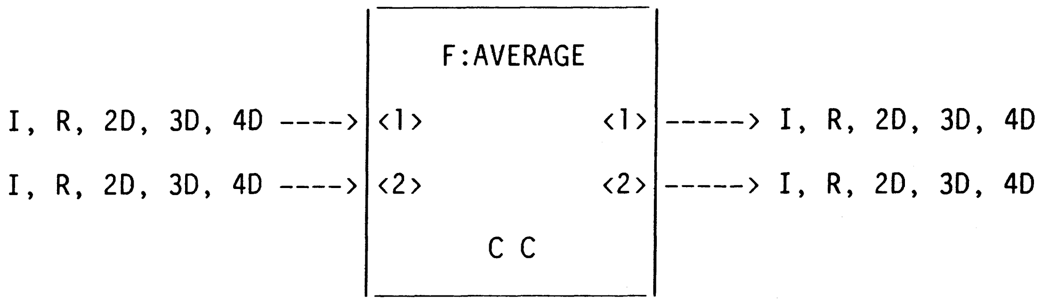
## NOTES

1. Each value on input <1> is scaled by the value on input <2>, then added to the internally stored current sum of scaled input <1> values. When the accumulated sum differs from the last value sent out output <1> by at least the amount on input <3>, the accumulated sum is output and the internal accumulated sum is cleared.

NOTES (continued)

2. If vectors are input on input <1>, the difference on input <3> is taken to be vector length. Vector length is the linear distance from a vector location to the origin of the world coordinate system (i.e., the Euclidean norm, $n(x,y) = x^2 + y^2$).

3. Sending a Boolean (TRUE or FALSE) to input <1> forces the accumulated sum to be output and cleared from internal storage.


F:AVERAGE
I, R, 2D, 3D, 4D ----> <1>
I, R, 2D, 3D, 4D ----> <2>
<1> -----> I, R, 2D, 3D, 4D
<2> -----> I, R, 2D, 3D, 4D
C C

## PURPOSE

Accepts two inputs, outputs the average of the two inputs on output <1>, and outputs the value of input <2> unchanged on output <2>.

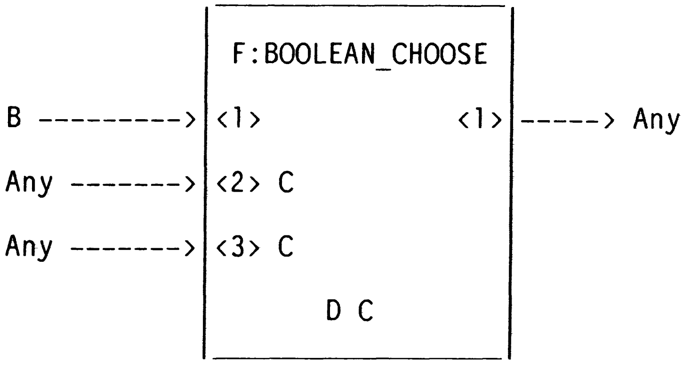
## DESCRIPTION

INPUT
<1> - any value
<2> - any value

OUTPUT
<1> - average of the two input values
<2> - value of input <2> unchanged

## NOTES

The two input values must be of the same data type (except a combination of real and integer is allowed); the outputs are also of that data type. If an integer is averaged with a real number, a real number is output on output <1>.



## PURPOSE

This function uses the Boolean on input <1> to select the constant message on input <2> or input <3>, outputting the selected message on output <1>.

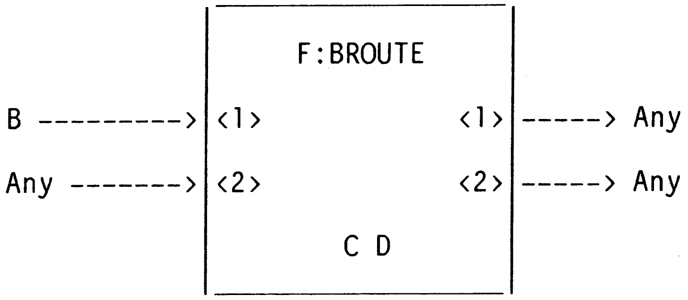
## DESCRIPTION

INPUT
<1> - Boolean
<2> - any message (constant)
<3> - any message (constant)

OUTPUT
<1> - message on input <2> or input <3>

## NOTES

A TRUE on input <1> selects the message on input <2>; a FALSE on input <1> selects the message on input <3>.



## PURPOSE

Acts as a Boolean route function, accepting a Boolean on input <1> and any message on input <2>. When a TRUE is received on input <1>, the message appears at output <1>. When a FALSE is received on input <1>, the message appears at output <2>.

## DESCRIPTION

INPUT
<1> - trigger
<2> - any message

OUTPUT
<1> - message on input <2> when input is <1> TRUE
<2> - message on input <2> when input is <1> FALSE

## ASSOCIATED FUNCTIONS

F:BROUTEC, F:CBROUTE

```
                 ____________________
                |                    |
                |     F:BROUTEC      |
                |                    |
B ------------> |<1>              <1>| -----> Any
                |                    |
Any ----------> |<2> C            <2>| -----> Any
                |                    |
                |        D D         |
                |____________________|
```

## PURPOSE

Acts as a Boolean route function, accepting a Boolean on input <1> and any message on constant input <2>. When a TRUE is received on input <1>, the message appears at output <1>. When a FALSE is received on input <1>, the message appears at output <2>.

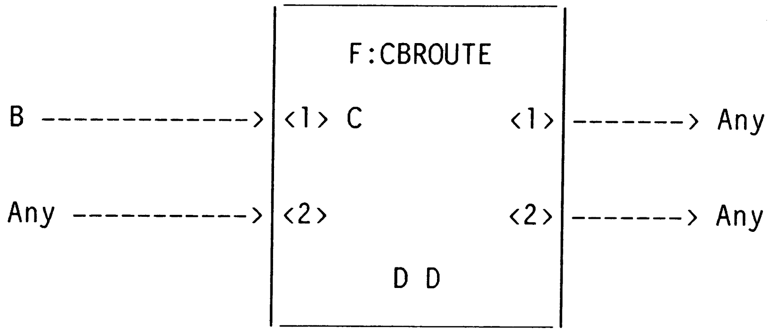
## DESCRIPTION

INPUT
<1> - trigger
<2> - any message (constant)

OUTPUT
<1> - message on input <2> when input is <1> TRUE
<2> - message on input <2> when input is <1> FALSE

## ASSOCIATED FUNCTIONS

F:BROUTE, F:CBROUTE



## PURPOSE

Acts as Boolean route function, sending the message on input <2> to output <1> when the Boolean on constant input <1> is TRUE or to output <2> when the constant Boolean on input <1> is FALSE.

## DESCRIPTION

INPUT
- <1> - trigger (constant)
- <2> - any message

OUTPUT
- <1> - message on input <2> when input <1> is TRUE
- <2> - message on input <2> when input <1> is FALSE

## ASSOCIATED FUNCTIONS

F:BROUTE, F:BROUTEC

```
                _______________________
               |                       |
               |    F:CCONCATENATE     |
S ------------>|<1> C             <1>  |-----> S
S ------------>|<2>               <2>  |-----> I
               |         D C           |
               |_______________________|
```

PURPOSE

Accepts two ASCII character strings and outputs on output <1> a string that is formed by concatenating the string on input <2> behind the string on input <1>. The length of the resulting string is sent on output <2>. Input <1> is a constant.

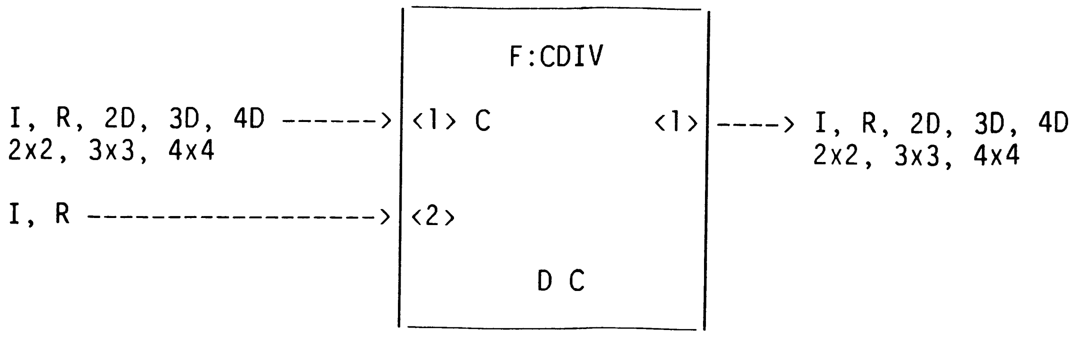
DESCRIPTION

INPUT
<1> - ASCII string (constant)
<2> - ASCII string

OUTPUT
<1> - concatenated string
<2> - length of the concatenated string

ASSOCIATED FUNCTIONS

F:CONCATENATE, F:CONCATENATEC



## PURPOSE

Accepts two inputs and produces an output that is the quotient of the two inputs (input <1> divided by input <2>). Input <1> is a constant.

## DESCRIPTION

INPUT
<1> - dividend (constant)
<2> - divisor

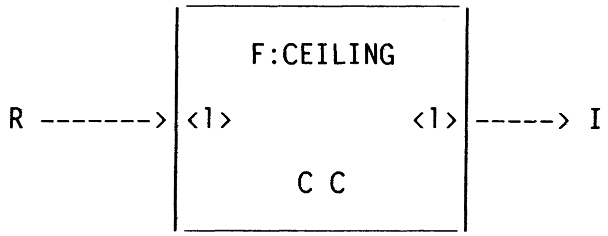
OUTPUT
<1> - quotient

## NOTES

The output is the same data type as input <1> (except when <1> is an integer and input <2> is a real; then a real is output). Input <2> should not be 0.

## ASSOCIATED FUNCTIONS

F:DIV, F:DIVC



## PURPOSE

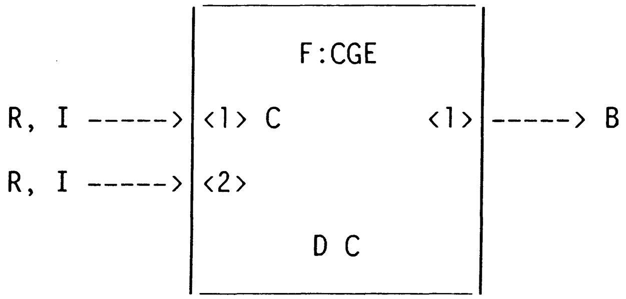
Rounds a real number away from zero to the nearest integer.

## DESCRIPTION

INPUT
<1> – real number to be rounded

OUTPUT
<1> – nearest integer

# F:CGE



## PURPOSE

Accepts any combination of reals and integers at its two inputs, and produces a Boolean output that is TRUE if input <1> is greater than or equal to input <2>, and FALSE otherwise. Input <1> is a constant.

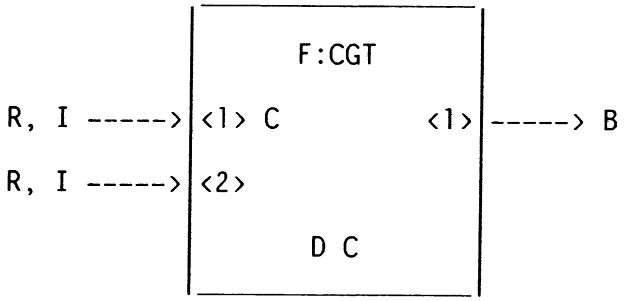
## DESCRIPTION

INPUT
<1> - real or integer to be compared (constant)
<2> - real or integer to be compared

OUTPUT
<1> - Boolean

## ASSOCIATED FUNCTIONS

F:GE, F:GEC



## PURPOSE

Accepts any combination of two reals or integers at its inputs, and produces a Boolean output that is TRUE if input <1> is greater than input <2>, and FALSE otherwise. Input <1> is a constant.

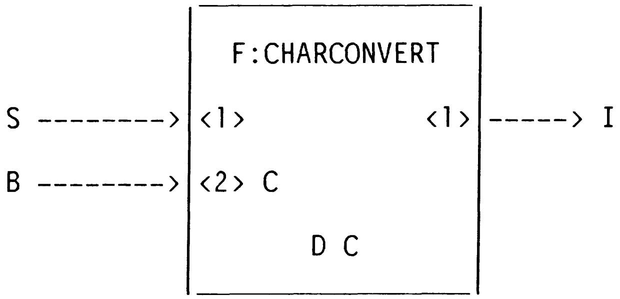
## DESCRIPTION

INPUT
<1> – real or integer to be compared (constant)
<2> – real or integer to be compared

OUTPUT
<1> – Boolean

## ASSOCIATED FUNCTIONS

F:GT, F:GTC



## PURPOSE

Converts the bytes of the string on input <1> into a stream of integers, one integer per byte.

## DESCRIPTION

INPUT
<1> - any string
<2> - Boolean (constant)

OUTPUT
<1> - stream of integers

## DEFAULTS

Boolean TRUE on input <2>.

## NOTES

1. The condition of the Boolean determines the range of bytes as integers as follows:

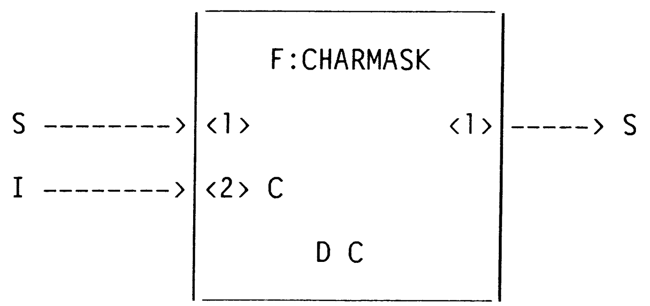
   TRUE = 0 to 255
   FALSE = -128 to 127 (2's complement)

NOTES (continued)

2. Note that if a TRUE is on input <2>, a value from 0-255 is output on <1>. If a FALSE is on input <2> and the value on input <1> is from 0-127, the value output is the same value that was input on <1>. If a FALSE is on input <2> and the value on input <1> is 128-255, a corresponding value between -128 and -1 is output.

EXAMPLE

'A' becomes 65
'AB' becomes 65 followed by 66



## PURPOSE

Masks each of the bytes of the string on input <1> by ANDing it with the integer on the constant input <2>, then outputs the masked string.

## DESCRIPTION

INPUT
<1> – any string
<2> – integer (constant)

OUTPUT
<1> – masked string

## NOTES

Only the low-order byte of the integer is used in the mask, i.e., integer 256 would be a 0 mask. Therefore, numbers between 0-255 are recommended.

```
                  ________________________
                 |                        |
                 |      F:CLCSECONDS      |
                 |                        |
I -------------->| <1> C             <1>  |---> I
                 |                        |
I -------------->| <2> C             <2>  |---> I
                 |                        |
B -------------->| <3> C             <3>  |---> B
                 |                        |
I -------------->| <4> C                  |
                 |                        |
I -------------->| <5> C                  |
                 |                        |
B -------------->| <6> C                  |
                 |                        |
                 |          D D           |
                 |________________________|
```

## PURPOSE

Generates outputs at timed intervals as specified by the inputs. All inputs to F:CLCSECONDS are constants. All outputs occur at the same timed interval. (Output <1> may be disabled.)

## DESCRIPTION

INPUT
- <1> - timed interval (constant)
- <2> - number of time intervals (constant)
- <3> - gate (constant)
- <4> - integer A (constant)
- <5> - integer B (constant)
- <6> - TRUE = run, FALSE = stop (constant)

OUTPUT
- <1> - integer A+B if input <3> is TRUE
- <2> - integer A+B
- <3> - TRUE if input <2> is not exceeded

(continued)

## NOTES

1. Input <1> is an integer that specifies a timed interval in hundredths of a second. Outputs from the function occur at this interval. Thus, a 10 on input <1> would specify a time interval of 1/10 second.

2. Input <2> is an integer that specifies the number of time intervals (duration) that the Boolean on output <3> will be TRUE. When this number of intervals is exceeded, the Boolean will be output as FALSE on each succeeding interval. Input <2> may be reset at any time, since the value at this input is decremented by 1 with each execution.

3. Input <3> is a Boolean that is used to gate the integer on output <1>. If the Boolean is TRUE, the integer (A+B) is output each timed interval. If the Boolean is FALSE, output <1> is disabled.

4. Inputs <4> and <5> are integers A and B, respectively. The sum of these integers is output as an integer on output <1> if the Boolean on input <3> is TRUE. This sum (A+B) is output as an integer on output <2>, independent of the condition of the Boolean on input <3>.

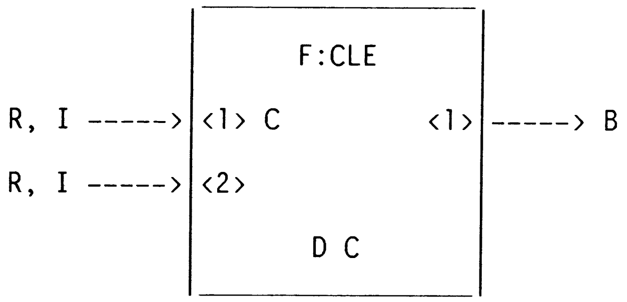
5. Input <6> is an optional switch. If input <6> receives no messages, the timer will run when there is a message on all of inputs <1> through <5>. If a Boolean FALSE is received on input <6>, the timer waits for a Boolean TRUE to be received on input <6> before running. No outputs are generated so long as input <6> is FALSE.

## ASSOCIATED FUNCTIONS

F:CLFRAMES, F:CLTICKS

## EXAMPLE

Refer to Application Notes 10 and 12 in the PS 300 Application Notes.



## PURPOSE

Accepts any combination of reals or integers at its inputs, and produces a Boolean output that is TRUE if input <1> is less than or equal to input <2>, and FALSE otherwise. Input <1> is a constant.

## DESCRIPTION

INPUT
<1> - value to be compared (constant)
<2> - value to be compared

OUTPUT
<1> - Boolean

## ASSOCIATED FUNCTIONS

F:LE, F:LEC

```
                    ________________________
                   |                        |
                   |      F:CLFRAMES        |
                   |                        |
I --------------->|<1> C              <1>|---> I
                   |                        |
I --------------->|<2> C              <2>|---> I
                   |                        |
B --------------->|<3> C              <3>|---> B
                   |                        |
I --------------->|<4> C                    |
                   |                        |
I --------------->|<5> C                    |
                   |                        |
B --------------->|<6> C                    |
                   |                        |
                   |          D D           |
                   |________________________|
```

## PURPOSE

Identical to F:CLCSECONDS and F:CLTICKS, except the time source is refresh frames.

## DESCRIPTION

INPUT

<1> - timed interval (constant)
<2> - number of time intervals (constant)
<3> - gate (constant)
<4> - integer A (constant)
<5> - integer B (constant)
<6> - TRUE = run, FALSE = stop (constant)

OUTPUT

<1> - A+B if input <3> is TRUE
<2> - A+B
<3> - TRUE if input <2> is not exceeded

(continued)

NOTES

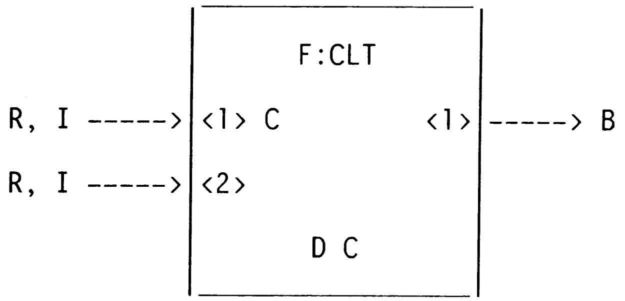
1. Input <1> is an integer that specifies a timed interval in frames. A frame is the length of time the Display Processor takes to draw the current structure once. The refresh rate is the number of frames per second. Outputs from the function occur at this interval.

2. Input <2> is an integer that specifies the number of timed intervals (duration) that the Boolean on output <3> will be TRUE. When this number of intervals is exceeded, the Boolean will be output as FALSE on each succeeding interval. Input <2> may be reset at any time.

3. Input <3> is a Boolean that is used to gate the integer on output <1>. If the Boolean is TRUE, the integer (A+B) is output each timed interval. If the Boolean is FALSE, output <1> is disabled.

4. Inputs <4> and <5> are integers A and B, respectively. The sum of these integers is output as an integer on output <1> if the Boolean on input <3> is TRUE. This sum (A+B) is output as an integer on output <2>, independent of the condition of the Boolean on input <3>.

5. Input <6> is an optional switch. If input <6> receives no messages, the timer will run when there is a message on all of inputs <1> through <5>. If a Boolean FALSE is received on input <6>, the timer waits for a Boolean TRUE to be received on input <6> before running. No outputs are generated so long as input <6> is FALSE.



## PURPOSE

Accepts any combination of reals or integers at its inputs, and produces a Boolean output that is TRUE if input <1> is less than input <2>, and FALSE otherwise. Input <1> is a constant.

## DESCRIPTION

INPUT
<1> – value to be compared (constant)
<2> – value to be compared

OUTPUT
<1> – Boolean

## ASSOCIATED FUNCTIONS

F:LT, F:LTC

```
                   ____________________________
                  |                            |
                  |         F:CLTICKS          |
I --------------> |<1> C                  <1> | ---> I
I --------------> |<2> C                  <2> | ---> I
B --------------> |<3> C                  <3> | ---> B
I --------------> |<4> C                       |
I --------------> |<5> C                       |
B --------------> |<6> C                       |
                  |            D D             |
                  |____________________________|
```

## PURPOSE

Identical to F:CLCSECONDS and F:CLFRAMES, except the time source is ticks of the 20 Hz system clock.

## DESCRIPTION

INPUT
<1> – timed interval (constant)
<2> – number of time intervals (constant)
<3> – gate (constant)
<4> – integer A (constant)
<5> – integer B (constant)
<6> – TRUE = run, FALSE = stop (constant)

OUTPUT
<1> – A+B if input <3> is TRUE
<2> – A+B
<3> – TRUE if input <2> is not exceeded

(continued)

NOTES

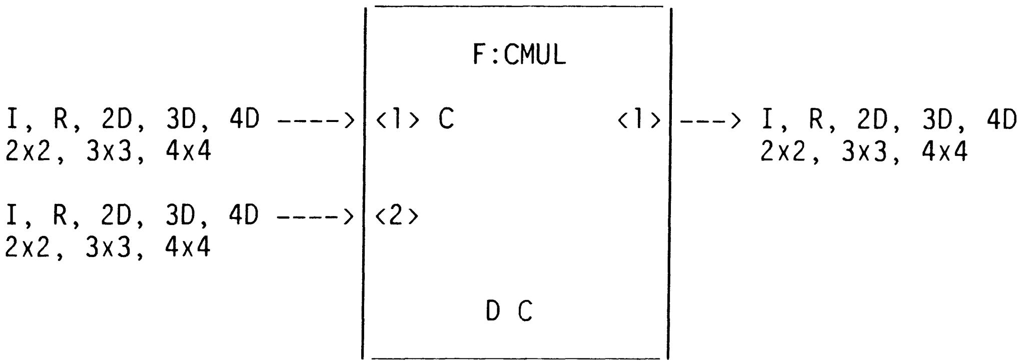
1. Input <1> is an integer that specifies a timed interval in ticks (where a tick is half the duration of the alternating current supply, 1/20 second in the U.S.). Outputs from the function occur at this interval.

2. Input <2> is an integer that specifies the number of timed intervals (duration) that the Boolean on output <3> will be TRUE. When this number of intervals is exceeded, the Boolean will be output as FALSE on each succeeding interval. Input <2> may be reset at any time.

3. Input <3> is a Boolean that is used to gate the integer output <1>. If the Boolean is TRUE, the integer (A+B) is output each timed interval. If the Boolean is FALSE, output <1> is disabled.

4. Inputs <4> and <5> are integers A and B, respectively. The sum of these integers is output as an integer on output <1> if the Boolean on input <3> is TRUE. This sum (A+B) is output as an integer on output <2>, independent of the condition of the Boolean on input <3>.

5. Input <6> is an optional switch. If input <6> receives no messages, the timer will run when there is a message on all of inputs <1> through <5>. If a Boolean FALSE is received on input <6>, the timer waits for a Boolean TRUE to be received on input <6> before running. No outputs are generated so long as input <6> is FALSE.



## PURPOSE

Accepts two inputs and outputs the product of the two inputs. Input <1> is a constant.

## DESCRIPTION

INPUT
<1> - any value (constant)
<2> - any value

OUTPUT
<1> - product

## NOTES

The two input values must be compatible data types; the output data type depends on the combination of input data types. Vectors are taken to be either row vectors (input <1>) or column vectors (input <2>).

## ASSOCIATED FUNCTIONS

F:MUL, F:MULC

# F:COLOR

```
                   -------------------
                  |                   |
                  |      F:COLOR      |
                  |                   |
2D, 3D ---------> |<1>             <1>| -----> 3D, 4D
                  |                   |
     R ---------> |<2>                |
                  |        C C        |
                  |___________________|
```

## PURPOSE

Accepts a 2D or 3D vector at input <1> and a real number representing a color-blended vector hue at input <2>, and outputs a 3D or 4D vector whose last "coordinate" is the hue value. This vector format is required for inputs to the vector list that has the color option specified.

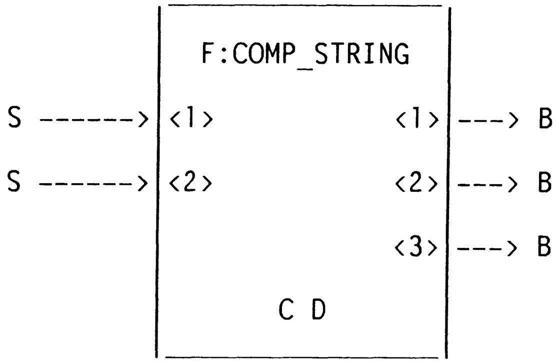
## DESCRIPTION

INPUT
<1> - vector
<2> - color-blended vector hue (0-720)

OUTPUT
<1> - vector whose last coordinate is the hue value

## NOTES

The real number at input <2> must be within the range 0-720; values outside this range are clamped to the nearest in-range value.



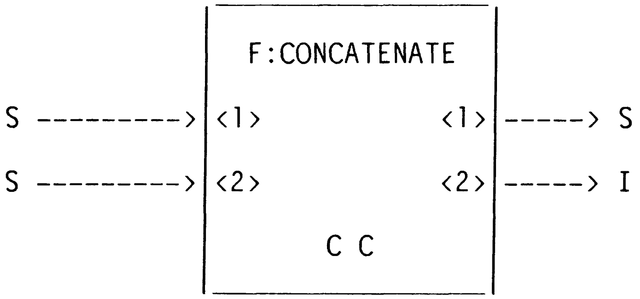
## PURPOSE

Compares two strings and sends a TRUE on output <1> if string 1 is less than string 2, a TRUE on output <2> if string 1 is equal to string 2, or a TRUE on output <3> if string 1 is greater than string 2.

## DESCRIPTION

INPUT
<1> - string
<2> - string

OUTPUT
<1> - TRUE = less than
<2> - TRUE = equal to
<3> - TRUE = greater than



## PURPOSE

Accepts two ASCII character strings and outputs a string that is formed by concatenating the string on input <2> behind the string on input <1>. The length of the resulting string is sent on output <2>.

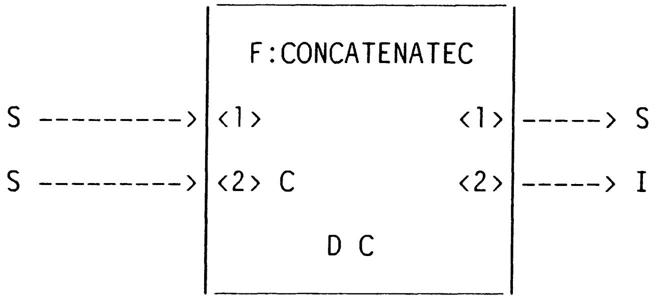
## DESCRIPTION

INPUT
<1> - ASCII string
<2> - ASCII string

OUTPUT
<1> - concatenated string
<2> - length of the concatenated string

## ASSOCIATED FUNCTIONS

F:CCONCATENATE, F:CONCATENATEC



## PURPOSE

Accepts two ASCII character strings and outputs a string that is formed by concatenating the string on input <2> behind the string on input <1>. The length of the concatenated string is sent on output <2>. Input <2> is a constant.

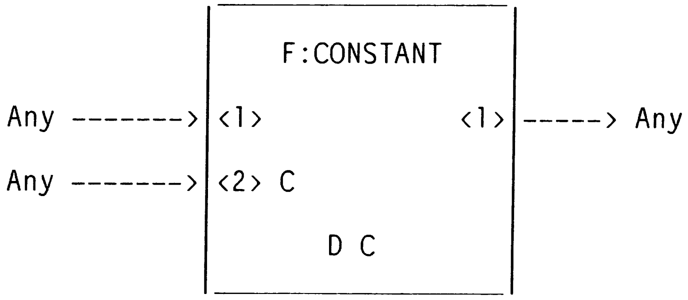
## DESCRIPTION

INPUT
<1> - ASCII string
<2> - ASCII string (constant)

OUTPUT
<1> - concatenated string
<2> - length of the concatenated string

## ASSOCIATED FUNCTIONS

F:CCONCATENATE, F:CONCATENATE



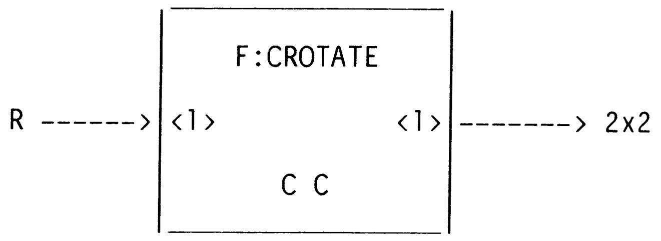
## PURPOSE

Accepts any message on inputs <1> and <2>. Input <2> is a constant. The constant message on input <2> is output on output <1> whenever a message is received on input <1>.

## DESCRIPTION

INPUT
<1> - trigger
<2> - any message (constant)

OUTPUT
<1> - message on input <2> when triggered



## PURPOSE

Creates a 2x2 Z rotation matrix.

## DESCRIPTION

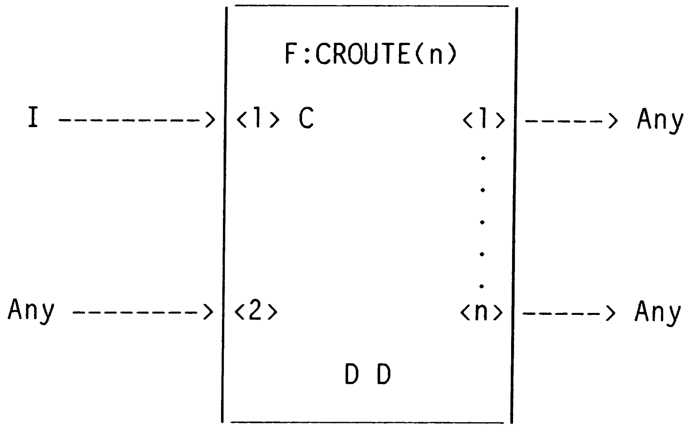
INPUT
<1> – degrees of rotation in Z

OUTPUT
<1> – 2x2 rotation matrix

## NOTES

1. The rotation matrix created by the function is normally used to update 2x2 matrix nodes in a display tree.

2. The "C" in the function's name stands for "character". 2x2 matrix nodes in display trees only affect character data nodes.



## PURPOSE

Accepts an integer on input <1> to switch the message on input <2> to the output specified by that integer. The message on input <2> may be of any data type. The integer on input <1> is a constant.

## DESCRIPTION

INPUT
<1> – integer (valid range 1 – 127) (constant)
<2> – any message

OUTPUT
<1> – message on input <2> when selected
.
.
<n> – message on input <2> when selected

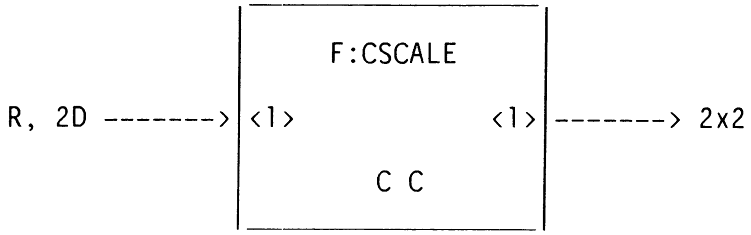
(continued)

## NOTES

The "n" in the function name may be any integer from 2 to 127. If the integer input is not a number from 1 to n, inclusive, then an error is detected and reported.

## ASSOCIATED FUNCTIONS

F:ROUTE(n), F:ROUTEC(n)



## PURPOSE

Scales characters. Accepts a real number or a 2D vector as a scaling factor for character strings. A 2x2 scaling matrix is output.

## DESCRIPTION

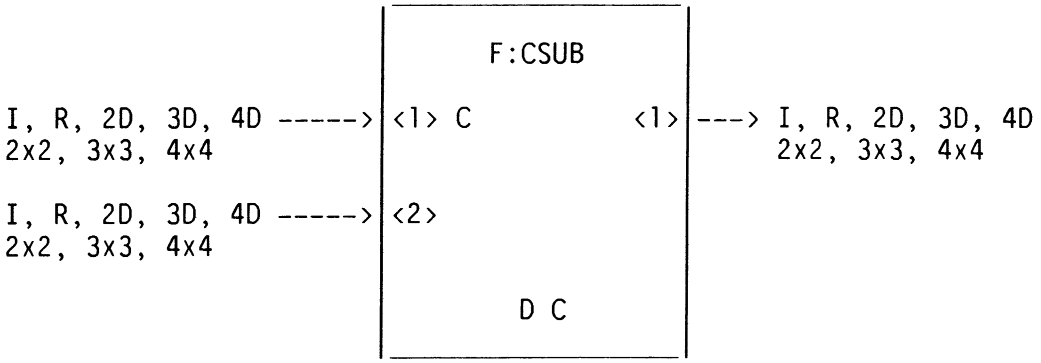
INPUT
<1> - scaling factor

OUTPUT
<1> - 2x2 scaling matrix

## NOTES

1. The scaling matrix is normally used to update a 2x2 matrix node in a display tree. The "C" in the function's name stands for "character". Only character data nodes are affected by 2x2 matrices.

2. If a real is input, the scaling factor represented by the real value is applied in X and Y. If a 2D vector is input, the X component of the vector is the scaling factor for X, and the Y component of the vector is the scaling factor for Y.



## PURPOSE

Accepts two inputs and produces an output that is the difference of the two inputs (input <2> is subtracted from input <1>). Input <1> is a constant.

## DESCRIPTION

INPUT
<1> – minuend (constant)
<2> – subtrahend

OUTPUT
<1> – difference

## NOTES

The two input values must be of the same data type (except a combination of real and integer is allowed); the output data type depends on the input data type(s).

## ASSOCIATED FUNCTIONS

F:SUB, F:SUBC

```
                           F:CVEC

R, 2D, 3D --------> <1> C               <1> -----> 2D, 3D, 4D

R ----------------> <2>

                             D C
```

## PURPOSE

Accepts two real numbers and outputs a 2D vector; accepts a 2D vector and a real number and outputs a 3D vector; or accepts a 3D vector and a real number and outputs a 4D vector.

## DESCRIPTION

INPUT
<1> - real, 2D, or 3D vector (constant)
<2> - real

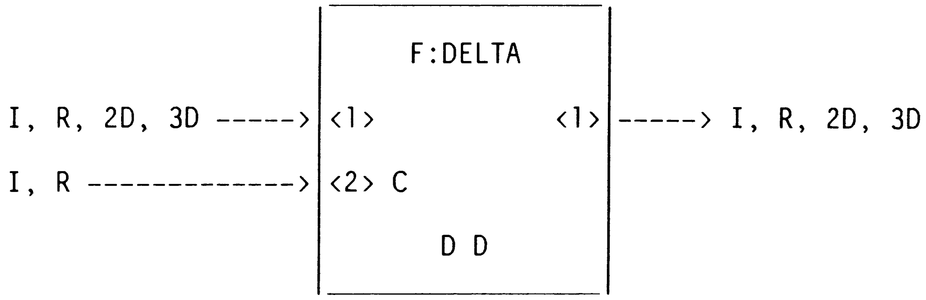
OUTPUT
<1> - 2D vector if input <1> is a real number
3D vector if input <1> is a 2D vector
4D vector if input <1> is a 3D vector

## NOTES

The output vector is the constant real number or vector from input <1> with the real number from input <2> appended as the last vector component.

## ASSOCIATED FUNCTIONS

F:VEC, F:VECC



## PURPOSE

Accepts integers, reals, 2D vectors, and 3D vectors on input <1> and integers or reals on input <2>. The value on input <1> is output on output <1> if it differs in magnitude from the previous input <1> value by at least the constant delta value on input <2>.

## DESCRIPTION

INPUT

<1> - integer, real, 2D, 3D vector

<2> - delta value (constant)

OUTPUT

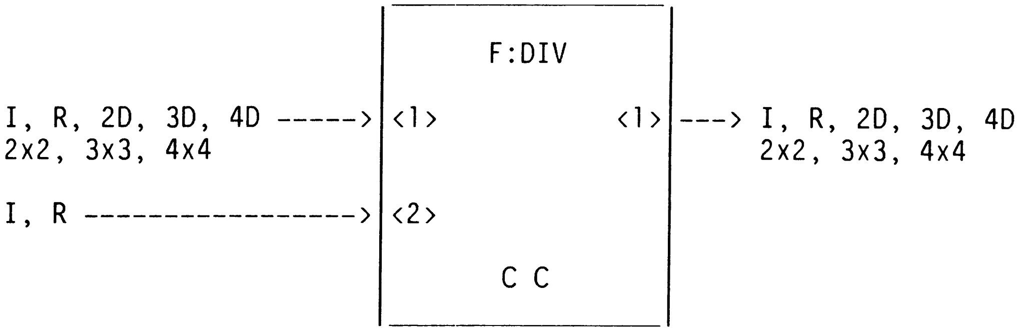
<1> - value on input <1> if it differs from the previous input <1> by at least the delta value on input <2>

## DEFAULTS

The first input <1> value is compared to 0 (zero).

## NOTES

The constant delta value on input <2> may be a real or an integer. If values on input <1> are reals or vectors the delta value on input <2> must be real. If input <1> is an integer, input <2> must also be an integer.



## PURPOSE

Accepts two inputs and produces an output that is the quotient of the two inputs (input <1> is divided by input <2>).

## DESCRIPTION

INPUT
<1> - dividend
<2> - divisor

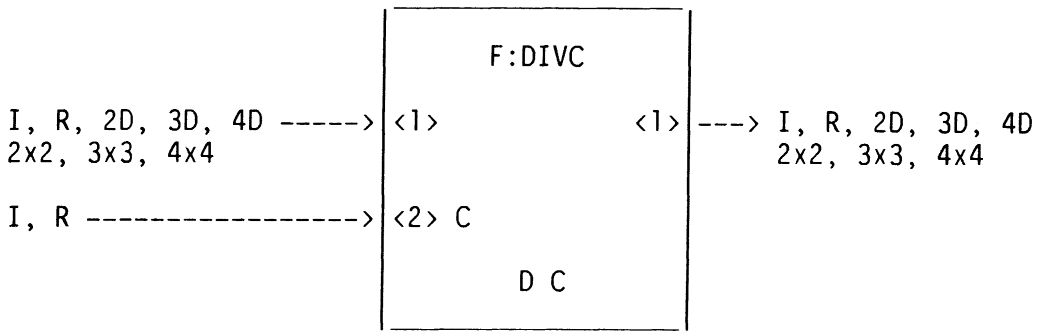
OUTPUT
<1> - quotient

## NOTES

The output is the same data type as input <1> (except when input <1> is an integer and input <2> is a real; then a real is output). Input <2> should not be 0.

## ASSOCIATED FUNCTIONS

F:DIVC, F:CDIV



## PURPOSE

Accepts two inputs and produces an output that is the quotient of the two inputs (input <1> is divided by input <2>). Input <2> is a constant.

## DESCRIPTION

INPUT
<1> – dividend
<2> – divisor (constant)

OUTPUT
<1> – quotient

## NOTES

The output is the same data type as input <1> (except when input <1> is an integer and input <2> is a real; then a real is output). Input <2> should not be 0.

## ASSOCIATED FUNCTIONS

F:DIV, F:CDIV

```
                   |          F:DSCALE          |
R ----------->     | <1>                    <1> | -----> 3x3
                   |                            |
R ----------->     | <2> C                  <2> | -----> R
                   |                            |
R ----------->     | <3> C                      |
                   |                            |
R ----------->     | <4> C                      |
                   |                            |
R ----------->     | <5> C                      |
                   |                            |
                   |            D C             |
```

## PURPOSE

Typically accepts real values originating from a control dial on input <1> and forms a 3x3 scaling matrix (output <1>) from the product of accumulated real values (input <1>) and the scaling factor on input <3>. Upper and lower scaling limits may be set on inputs <4> and <5>, respectively. If the accumulator content exceeds the upper limit (input <4>), then the upper limit value is sent out on output <1>. Likewise, if the product is below the lower limit, the lower limit value is sent out on output <1>.

## DESCRIPTION

INPUT

<1> - delta
<2> - accumulator set (constant)
<3> - scaling factor (constant)
<4> - upper limit (constant)
<5> - lower limit (constant)

OUTPUT

<1> - 3x3 scaling matrix
<2> - accumulator contents

## DEFAULTS

Inputs <3>, <4>, and <5> are optional. If input <3> receives no messages, a scaling factor of 1 is the default value. If inputs <4> and/or <5> receive no messages, no upper and/or lower limits are set.

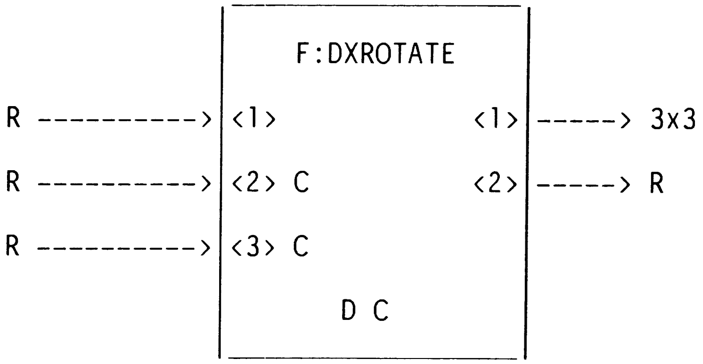
## NOTES

1. Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 1). The current accumulator content is output on output <2>.

2. It is sometimes valuable to limit the upper range of scaling to a value that will not cause data to overflow the viewport. Also, lower limits may be set to keep the object to a size that allows the object to be viewed easily and to prevent negative scaling.

## EXAMPLE

Refer to Application Note 6 in the PS 300 Application Notes.



## PURPOSE

Typically accepts real values originating from a control dial on input <1> and produces a 3x3 rotation matrix (output <1>) from the angle derived from the accumulated sum of the real values on input <1>, multiplied by the scale factor received on input <3>. Rotation is around the X axis.

## DESCRIPTION

INPUT
<1> - rotation delta
<2> - initial accumulator value (constant)
<3> - scale factor (constant)

OUTPUT
<1> - 3x3 rotation matrix in X
<2> - current accumulator value

## DEFAULTS

If input <3> receives no messages, a scale factor of 1 is the default value.

## NOTES

Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 0). The current accumulator value is output on output <2>.

```
                  -------------------------
                 |                         |
                 |        F:DYROTATE       |
                 |                         |
R ------------>  |<1>                   <1>| -----> 3x3
                 |                         |
R ------------>  |<2> C                 <2>| -----> R
                 |                         |
R ------------>  |<3> C                    |
                 |                         |
                 |           D C           |
                  -------------------------
```

## PURPOSE

Typically accepts real values originating from a control dial on input <1> and produces a 3x3 rotation matrix (output <1>) from the angle derived from the accumulated sum of the real values on input <1>, multiplied by the scale factor received on input <3>. Rotation is around the Y axis.

## DESCRIPTION

INPUT
<1> - rotation delta
<2> - initial accumulator value (constant)
<3> - scale factor (constant)

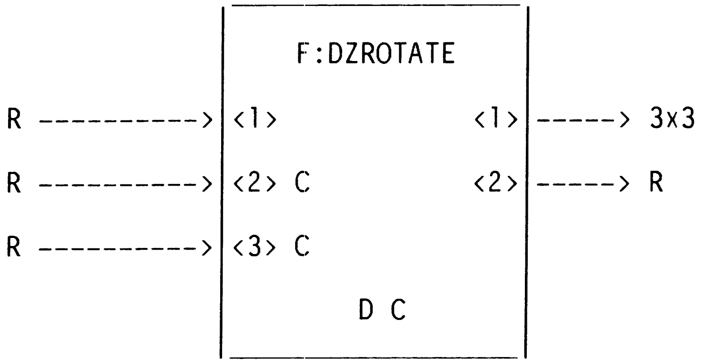
OUTPUT
<1> - 3x3 rotation matrix in Y
<2> - current accumulator value

## DEFAULTS

If input <3> receives no messages, a scale factor of 1 is the default value.

## NOTES

Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 0). The current accumulator value is output on output <2>.



## PURPOSE

Typically accepts real values originating from a control dial on input <1> and produces a 3x3 rotation matrix (output <1>) from the angle derived from the accumulated sum of the real values on input <1>, multiplied by the scale factor received on input <3>. Rotation is around the Z axis.

## DESCRIPTION

INPUT

<1> - rotation delta
<2> - initial accumulator value (constant)
<3> - scale factor (constant)

OUTPUT

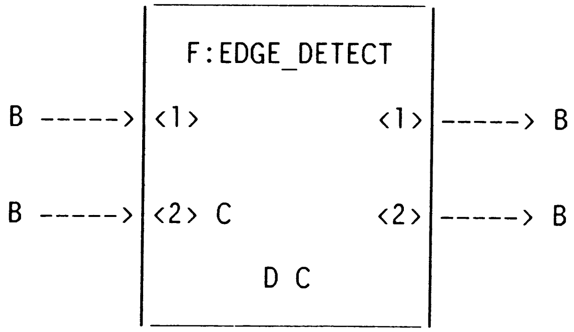
<1> - 3x3 rotation matrix in Z
<2> - current accumulator value

## DEFAULTS

If input <3> receives no messages, a scale factor of 1 is the default value.

## NOTES

Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 0). The current accumulator content is output on output <2>.



## PURPOSE

Accepts Boolean values on inputs <1> and <2>. Input <2> is a constant. Whenever the state of the Boolean on input <1> changes to match the state on input <2>, the Boolean on input <1> is output on output <1>, and the complement of that value is output on output <2>.

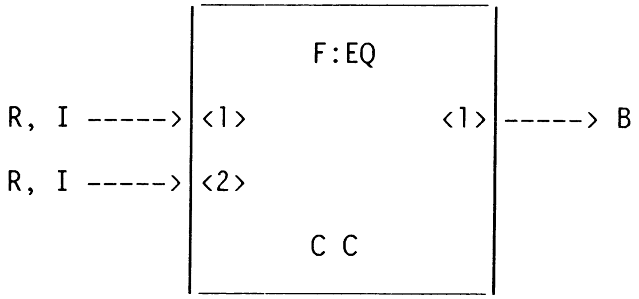
## DESCRIPTION

INPUT
<1> - Boolean
<2> - Boolean (constant)

OUTPUT
<1> - Boolean on input <1> when this matches input <2>
<2> - complement of output <1>

## NOTES

By connecting output <2> to input <2>, all transitions are detected.



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> equals input <2>, and FALSE otherwise.

## DESCRIPTION

INPUT
<1> - real or integer to be compared
<2> - real or integer to be compared

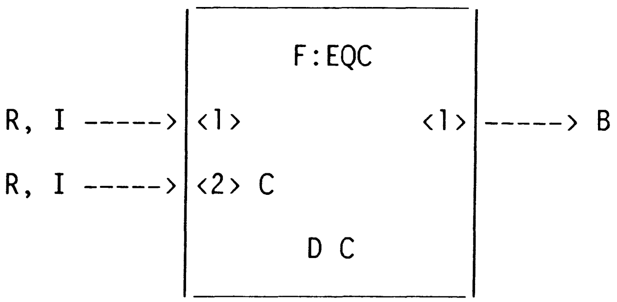
OUTPUT
<1> - TRUE if input <1> equals input <2>, else FALSE

## NOTES

Inputs do not have to be of the same data type.

## ASSOCIATED FUNCTIONS

F:EQC



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> equals input <2>, and FALSE otherwise. Input <2> is a constant.

## DESCRIPTION

INPUT
<1> – real or integer to be compared
<2> – real or integer to be compared (constant)

OUTPUT
<1> – TRUE if input <1> equals input <2>, else FALSE

## NOTES

Inputs do not have to be of the same data type.

## ASSOCIATED FUNCTIONS

F:EQ

# F:FETCH

Intrinsic Function
Miscellaneous

```
               ____________________
              |                    |
              |      F:FETCH       |
Any -----> |<1>             <1>|-----> Any
              |                    |
S -------> |<2> C               |
              |                    |
              |        D C         |
              |____________________|
```

## PURPOSE

Accepts a string which is the name of a variable on input <2>. When any message is received on input <1>, the message currently stored in the variable named on input <2> is fetched and output from this function. The message stored in the named variable may be of any data type. The arrival of input <1> is used to activate the function, but is otherwise ignored. Input <2> is a constant.

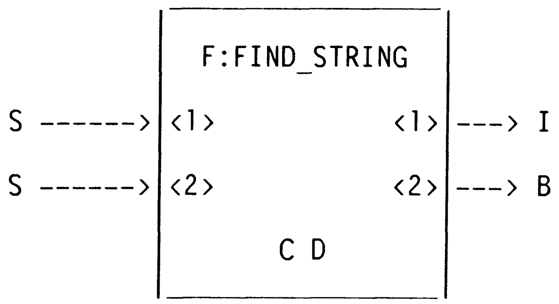
## DESCRIPTION

INPUT

<1> - trigger
<2> - variable name (constant)

OUTPUT

<1> - message associated with variable name on input <2>



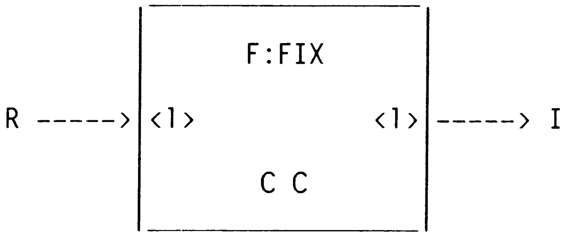
## PURPOSE

If the string on input <2> is a substring of the string on input <1>, the starting position of the substring and a Boolean TRUE are output. A FALSE is output if the substring cannot be found and nothing is sent on output <1>.

## DESCRIPTION

INPUT
<1> - string
<2> - substring

OUTPUT
<1> - starting position of the substring, if found
<2> - TRUE = substring found, FALSE = not found



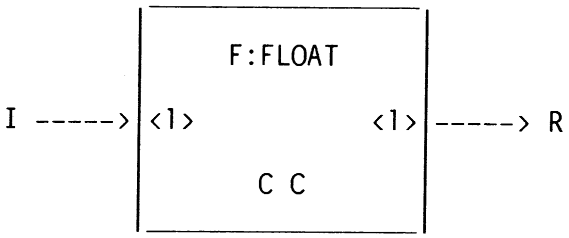
## PURPOSE

Accepts a real number and outputs a value that is truncated to an integer (toward zero).

## DESCRIPTION

INPUT
<1> – real number

OUTPUT
<1> – real on input <1> truncated to an integer



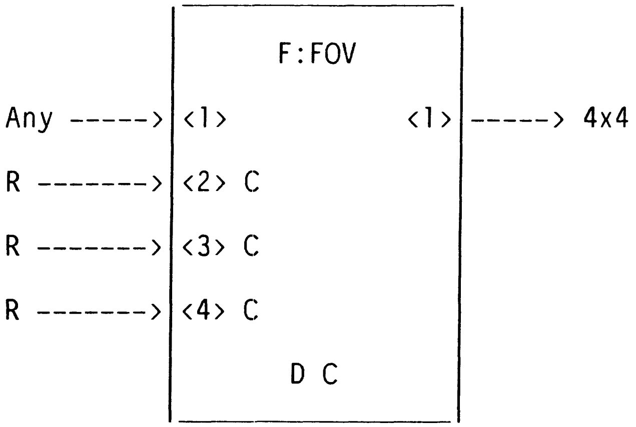
## PURPOSE

Accepts an integer and outputs a real number of the same value.

## DESCRIPTION

INPUT
<1> - integer

OUTPUT
<1> - real number of the same value as input <1>



PURPOSE

This is the functional counterpart of the FIELD_OF_VIEW command. The field of view that is specified by this function is used for perspective projections.

DESCRIPTION

INPUT
<1> - trigger
<2> - viewing angle (constant)
<3> - front boundary (constant)
<4> - back boundary (constant)

OUTPUT
<1> - 4x4 matrix

## NOTES

1. The message on input <1> acts as a trigger to the function.

2. The constant real value on input <2> represents the viewing angle in degrees. This angle defines the viewing frustum.

3. The front boundary and back boundary of the viewing frustum are specified as constant real numbers on inputs <3> and <4>, respectively.

4. The field of view specified on the inputs to F:FOV is output as a 4x4 matrix.

## ASSOCIATED FUNCTIONS

F:WINDOW, F:MATRIX_4X4

```
              | F:GATHER_STRING       |
S ------>     | <1>              <1>  |---> S
CH ----->     | <2> C            <2>  |---> I
B ------>     | <3> C                 |
              |          D C          |
```

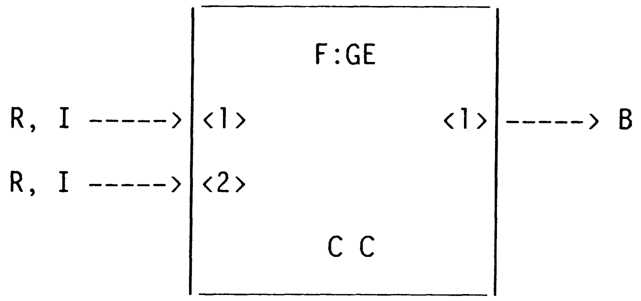
## PURPOSE

Collects strings that arrive at input <1> until the terminator character on input <2> arrives. Concatenates all strings into one packet and outputs the concatenated string on output <1>. If the Boolean on input <3> is TRUE, the terminator character is appended to the string. Output <2> contains the length of the string. Inputs <2> and <3> are constants.

## DESCRIPTION

INPUT
<1> – string
<2> – packet terminator (constant)
<3> – TRUE = with terminator, FALSE = without terminator (constant)

OUTPUT
<1> – concatenated string (packet)
<2> – length of the string



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is greater than or equal to input <2>, and FALSE otherwise.

## DESCRIPTION

INPUT

<1> - value to be compared
<2> - value to be compared

OUTPUT

<1> - TRUE if input <1> is greater than or equal to input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:GEC, F:CGE

# F:GEC

```
                 ____________________
                |                    |
                |       F:GEC        |
                |                    |
R, I ----->     |<1>              <1>|-----> B
                |                    |
R, I ----->     |<2> C               |
                |                    |
                |        D C         |
                |____________________|
```

## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is greater than or equal to input <2>, and FALSE otherwise. Input <2> is a constant.

## DESCRIPTION

INPUT

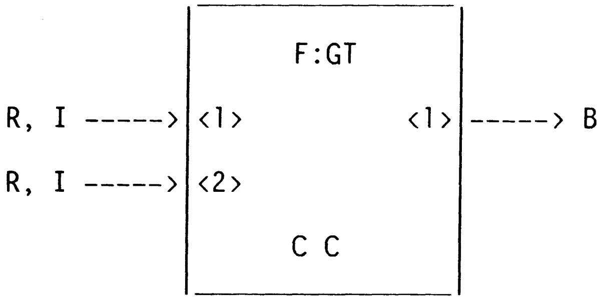
<1> - value to be compared
<2> - value to be compared (constant)

OUTPUT

<1> - TRUE if input <1> is greater than or equal to input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:GE, F:CGE



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is greater than input <2>, and FALSE otherwise.

## DESCRIPTION

INPUT
<1> – value to be compared
<2> – value to be compared

OUTPUT
<1> – TRUE if input <1> greater than input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:GTC, F:CGT

```
                 ____________________
                |                    |
                |       F:GTC        |
                |                    |
R, I ----->     |<1>              <1>|-----> B
                |                    |
R, I ----->     |<2> C               |
                |                    |
                |        C C         |
                |____________________|
```

## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is greater than input <2>, and FALSE otherwise. Input <2> is a constant.

## DESCRIPTION

INPUT

<1> - value to be compared
<2> - value to be compared (constant)

OUTPUT

<1> - TRUE if input <1> greater than input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:GT, F:CGT

# F:INPUTS_CHOOSE(n)

```
                   ____________________
                  |                    |
                  | F:INPUTS_CHOOSE(n) |
                  |                    |
Any ------------> |<1> C            <1>| -------> Any
 .                | .                  |
 .                | .                  |
 .                | .                  |
Any ------------> |<n-1> C             |
                  |                    |
I --------------> |<n>                 |
                  |                    |
                  |        D C         |
                  |____________________|
```

## PURPOSE

Accepts an integer with a value from 1 to (n-1) on input <n> and uses that value to choose which of inputs <1> through <n-1> to accept as an input. The chosen message is then output.

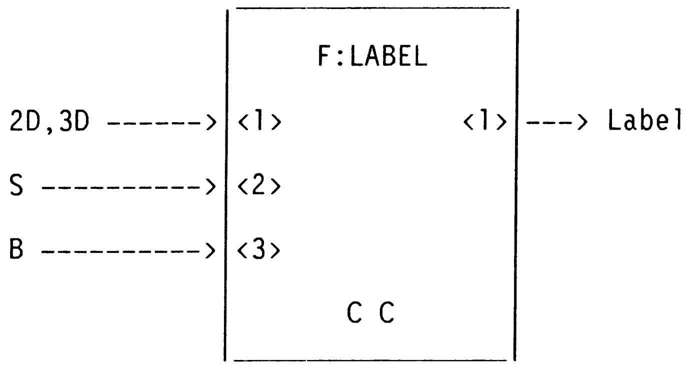
## DESCRIPTION

INPUT
<1> - any message (constant)
<n-1>- any message (constant)
<n> - chosen message number

OUTPUT
<1> - chosen message

## NOTES

To set up F:INPUTS_CHOOSE(n) for a given number of messages between 2 and 127 inclusive, add one to the number of messages and substitute the result for "n" in the function identifier. For example, F:INPUTS_CHOOSE(5) accepts four messages at inputs <1> through <4>. The selector input is always input <n>. Thus, for F:INPUTS_CHOOSE(5), the selector input is <5>.



PURPOSE

Creates a label to send to a labels node using the vector on input <1> as the position of the label and the string on input <2> as the text of the label. Input <3> indicates whether the the label is displayed or not.

DESCRIPTION

INPUT
<1> - X, Y, and (optionally) Z location of the label
<2> - text of the label
<3> - TRUE = displayed, FALSE = not displayed

OUTPUT
<1> - label for input to a labels node

NOTES

1. The data type output by this function can only be used to update a labels node. It is not accessible or printable.

```
            ________________________
           |                        |
           |     F:LBL_EXTRACT      |
           |                        |
I ------>  |<1>                  <1>|---> I
           |                        |
S ------>  |<2>                  <2>|---> 2D, 3D, 4D
           |                        |
           |                     <3>|---> S
           |                        |
           |                     <4>|---> B
           |                        |
           |          C C           |
           |________________________|
```

## PURPOSE

Extracts information about a string from a **LABELS** node given an index into the labels block on input <1> and the name of the labels node on input <2>.

## DESCRIPTION

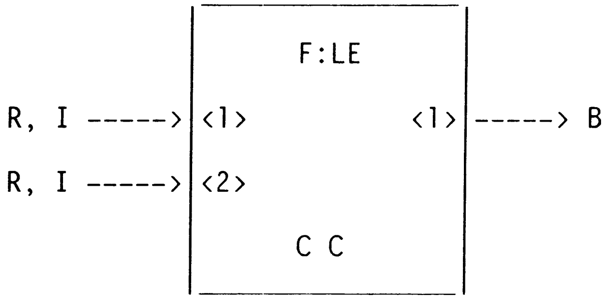
INPUT
- <1> - index of the string in question
- <2> - name of the **LABELS** node

OUTPUT
- <1> - data type
- <2> - the start location of the string in question
- <3> - the text of the string
- <4> - TRUE = on, FALSE = off

## NOTES

1. The integer on output <1> is the same as would be sent from output <7> of F:PICKINFO.

2. Output <4> indicates whether the string is on or off.



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is less than or equal to input <2>, and FALSE otherwise.

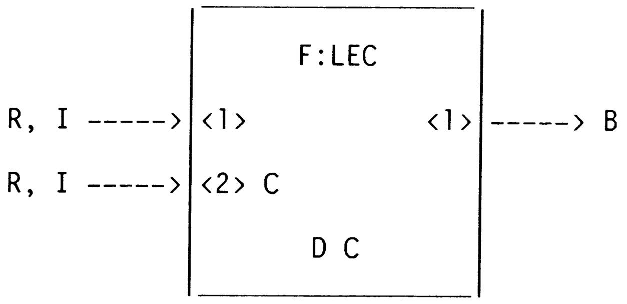
## DESCRIPTION

INPUT
<1> - value to be compared
<2> - value to be compared

OUTPUT
<1> - TRUE if input <1> is less than or equal to input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:LEC, F:CLE



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is less than or equal to input <2>, and FALSE otherwise. Input <2> is a constant.

## DESCRIPTION

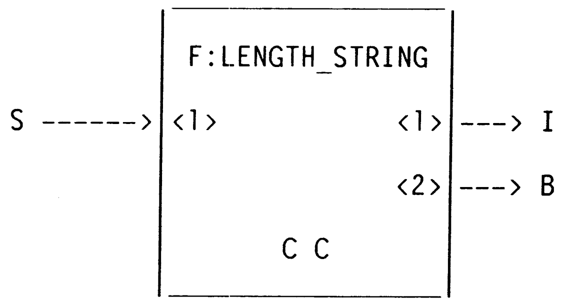
INPUT
<1> - value to be compared
<2> - value to be compared (constant)

OUTPUT
<1> - TRUE if input <1> is less than or equal to input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:LE, F:CLE

# F:LENGTH_STRING



## PURPOSE

Outputs the length of a string.

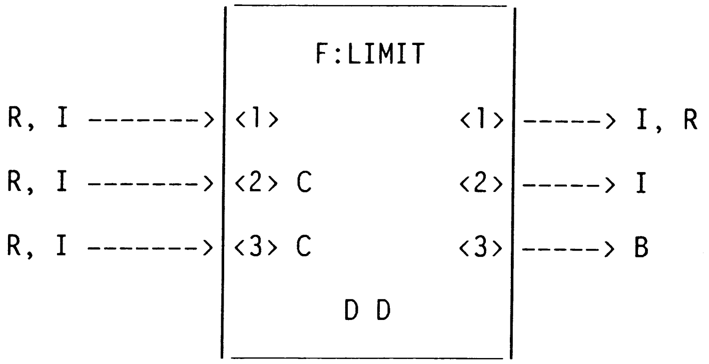
## DESCRIPTION

INPUT
<1> - string

OUTPUT
<1> - length of the string
<2> - TRUE = null string, FALSE otherwise

## NOTES

1. A possible output is zero.



## PURPOSE

Accepts real number or integer values on all inputs; all three input values must be of the same data type. The output data type is the same as the input data type.

## DESCRIPTION

INPUT
<1> - value
<2> - upper limit (constant)
<3> - lower limit (constant)

OUTPUT
<1> - input <1> if this value is in range
<2> - in-range value
<3> - TRUE if in-range, FALSE if out-of-range

## NOTES

1. The value on input <1> is compared to the constant upper limit value on input <2> and the constant lower limit value on input <3>.

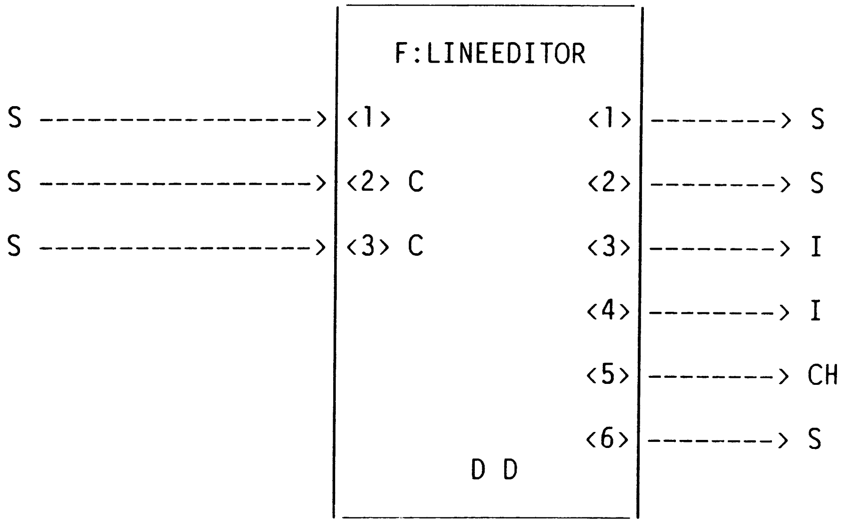
2. If the input <1> value is in range, that value is output unchanged on output <1> and output <2>, and a TRUE is output on output <3>.

(continued)

NOTES (continued)

3. If the input <1> value is out of range, the output <1> value is adjusted to the nearer limit (as set by inputs <2> and <3>), output <2> is disabled, and output <3> is FALSE.

4. If the value on input <2> is less than or equal to the value on input <3>, the function will always output the value received on input <3>.



## PURPOSE

Accepts a stream of characters and simple editing commands, accumulates the characters in an internal line buffer, applies the commands to the contents of the line buffer as they are received, and outputs the edited line when a specified delimiter character is recognized.

## DESCRIPTION

INPUT
<1> – editing commands and material to be edited (input string)
<2> – prompt message (constant)
<3> – line delimiter (constant)

OUTPUT
<1> – edited output
<2> – display output
<3> – integer for <clear> of CHARACTERS
<4> – integer for <append> of CHARACTERS
<5> – character for <append> of CHARACTERS
<6> – string for <substitute> or <replace> of CHARACTERS

(continued)

NOTES

1. In a typical application, **F:LINEEDITOR** receives its input from the PS 300 keyboard and sends its edited output either to a terminal (such as the debug terminal or the Terminal Emulator) or to a **CHARACTERS** node in the PS 300 display tree. A specially-formatted "display" output is used for terminals; other outputs are intended as connections into **CHARACTERS** to allow keyboard editing of a **CHARACTERS** string.

2. **F:LINEEDITOR** recognizes the following editing commands:

   Delete (Hex '7F'): Deletes the most recently received character from the internal line buffer.

   Control-U (Hex '15'): Deletes the entire line buffer. Redisplays a predetermined prompt message at any associated terminals by sending the prompt string on the display output <2>.

   Control-R (Hex '12'): Retypes the entire line (preceded by the prompt message) at any associated terminals by sending the prompt and line along the display output <2>.

3. Input <1> receives the stream of strings to be collected and edited, along with all editing commands. The PS 300 keyboard is typically connected to this input.

4. Input <2> contains a prompt message, if one is needed. The prompt string may contain one or several characters. This prompt appears only at output <2>, and it appears there whenever a control-U, a control-R, or a delimiter is received at input <1>. The prompt message is optional and there is no default.

5. Input <3> contains a single character designated as the delimiter. When this character is received at input <1>, the contents of the line buffer appear at outputs <1> and <6> (edited by the editing commands), and at output <2> (along with the prompt).

6. The default delimiter is <cr> (carriage-return; Hex '0D'), but this <cr> is always expanded to <cr><lf> (carriage-return/line-feed; Hex '0D0A') for output at <1>, <2>, and <6>.

NOTES (continued)

7. If input <3> contains a non-<cr> delimiter <delim>, this delimiter is passed on as is to outputs <1> and <6>, but it is always converted to <delim><cr><lf> for output <2> (the display output). (This implies that specifying a delimiter of <lf> produces double-spaced display output.)

8. Output <1> contains the contents of the line buffer, which in turn is composed of the collected and edited characters from input <1>. This output fires when a delimiter is recognized at input <1> or when 255 characters have been collected since the last firing or since initialization.

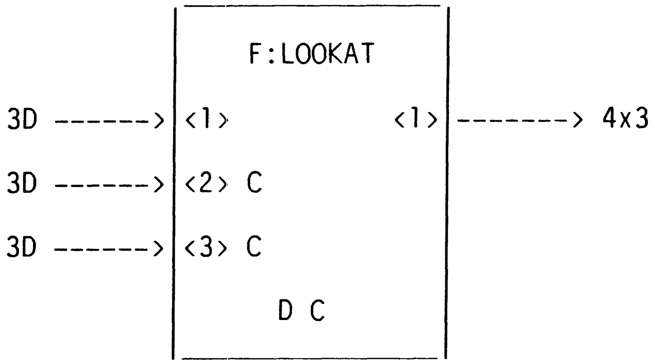
9. Output <2> is the display output. Unlike outputs <1> and <6>, this output includes "editing effects" intended for terminal display (prompt messages, displayed Control-U's and Control-R's, character erasures corresponding to deletes, and so on). For the treatment of delimiters at output <2>, see note 7 above.

10. Output <3> is an integer output intended as a connection into the <clear> input of a CHARACTERS command. The integer is sent whenever a control-U is received at input <1>.

11. Output <4> always sends an integer 1, and is intended as a connection into the <delete> input of a CHARACTERS command. The 1 is sent whenever a delete is received at input <1>.

12. Output <5> is intended as a connection into the <append> input of a CHARACTERS command. This output passes on all characters received at input <1> except editing commands (delete, control-U, control-R). No buffering is performed at this output -- it fires once for each non-command character, and the message is always a single character.

13. Output <6> is intended as a connection into the <substitute> or <replace> input of a CHARACTERS command. It fires whenever the function is activated by a (single-character or multi-character) string at input <1>. In addition, output <6> fires whenever output <1> fires.



## PURPOSE

Accepts three 3D vectors that specify the position to "look at", the position to "look from", and which direction is "up". Inputs <2> and <3> ("look from" and "up" orientation) are constants.

## DESCRIPTION

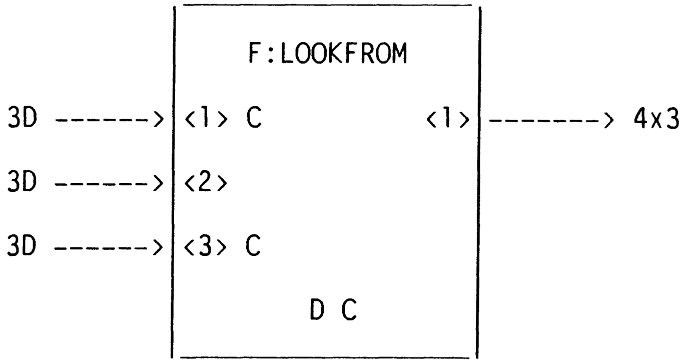
INPUT
<1> - look at point
<2> - look from point (constant)
<3> - up orientation (constant)

OUTPUT
<1> - 4x3 viewing matrix

## NOTES

1. Input <1>, the "look at" vector, triggers the function.

2. The 3D vectors are used to generate a 4x3 matrix that may be used to update a LOOK viewing transformation node in a display tree.



## PURPOSE

Accepts three 3D vectors that specify the position to "look at", the position to "look from", and which direction is "up". Inputs <1> and <3> ("look at" and "up" orientation) are constants.

## DESCRIPTION

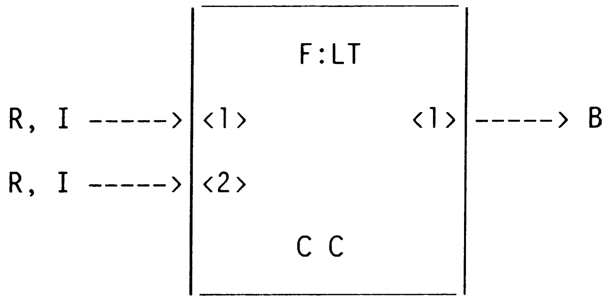
INPUT
<1> – look at point (constant)
<2> – look from point
<3> – up orientation (constant)

OUTPUT
<1> – 4x3 viewing matrix

## NOTES

1. Input <2>, the "look from" vector, triggers the function.

2. The 3D vectors are used to generate a 4x3 matrix that may be used to update a LOOK viewing transformation node in a display tree.



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is less than input <2>, and FALSE otherwise.

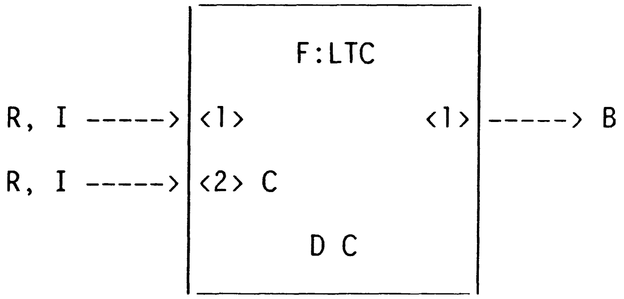
## DESCRIPTION

INPUT
<1> - value to be compared
<2> - value to be compared

OUTPUT
<1> - TRUE if input <1> is less than input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:LTC, F:CLT



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is less than input <2>, and FALSE otherwise. Input <2> is a constant.

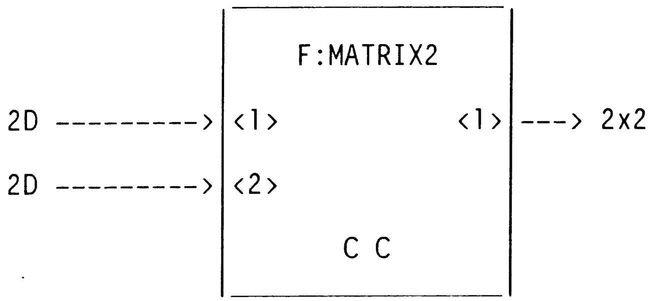
## DESCRIPTION

INPUT
<1> - value to be compared
<2> - value to be compared (constant)

OUTPUT
<1> - TRUE if input <1> is less than input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:LT, F:CLT



PURPOSE

Accepts two 2D vectors and produces a 2x2 matrix.

DESCRIPTION

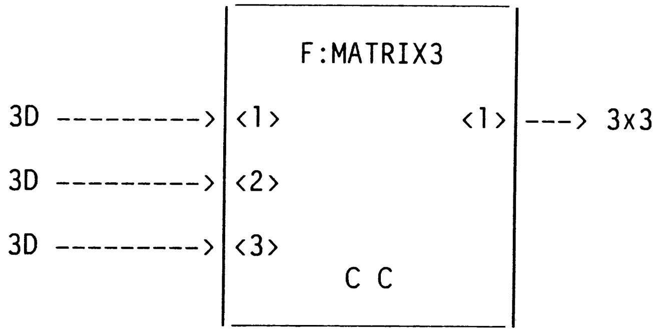
INPUT
<1> - 2D vector
<2> - 2D vector

OUTPUT
<1> - 2x2 matrix

NOTES

1. The matrix output may be used to update a 2x2 matrix node in a display tree or as input to another function.

2. The vector on input <1> is output as the first row of the matrix. The vector on input <2> is output as the second row.



## PURPOSE

Accepts three 3D vectors and produces a 3x3 matrix.

## DESCRIPTION

INPUT
<1> - 3D vector
<2> - 3D vector
<3> - 3D vector

OUTPUT
<1> - 3x3 matrix

## NOTES

1. The matrix output may be used to update a 3x3 matrix node in a display tree or as input to another function.

2. The vector on input <1> is output as the first row of the matrix. The vector on input <2> is output as the second row. The vector on input <3> is the third row.

```
                |---------------------|
                |      F:MATRIX4      |
4D ---------->  |<1>               <1>|---> 4x4
4D ---------->  |<2>                  |
4D ---------->  |<3>                  |
4D ---------->  |<4>                  |
                |         C C         |
                |---------------------|
```

## PURPOSE

Accepts four 4D vectors and produces a 4x4 matrix.

## DESCRIPTION

INPUT
<1> – 4D vector
<2> – 4D vector
<3> – 4D vector
<4> – 4D vector

OUTPUT
<1> – 4x4 matrix

## NOTES

1. The matrix output may be used to update a 4x4 matrix node in a display tree or as input to another function.

2. The vector on input <1> is output as the first row of the matrix. The vector on input <2> is output as the second row. The vector on input <3> is the third row. The vector on input <4> is the fourth row.

```
              ______________________
             |                      |
             |  F:MCONCATENATE(n)   |
             |                      |
S ------>    | <1>              <1> | ---> S
             |  .                   |
             |  .                   |
S ------>    | <n>              <2> | ---> I
             |                      |
             |         C C          |
             |______________________|
```

PURPOSE

Accepts strings on inputs <1> through <n> and concatenates them into a single string. Output <1> contains the resulting string and output <2> contains its length.

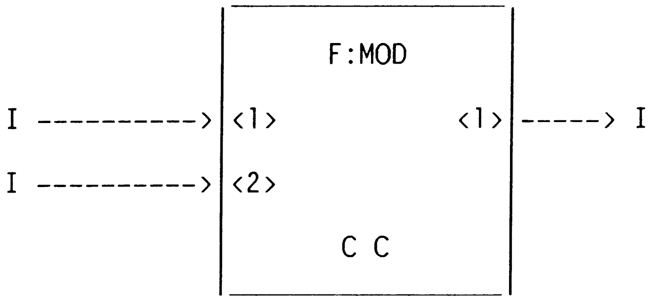
DESCRIPTION

INPUT
<1> - string
<n> - string

OUTPUT
<1> - concatenated string
<2> - string length

NOTES

1. The limit to the number of inputs to this function is 127.



## PURPOSE

Accepts two integers as inputs and produces an integer output that is the remainder resulting from the division of the value on input <1> by the value on input <2>. The integer on input <2> must be positive.

## DESCRIPTION

INPUT
<1> - integer
<2> - integer

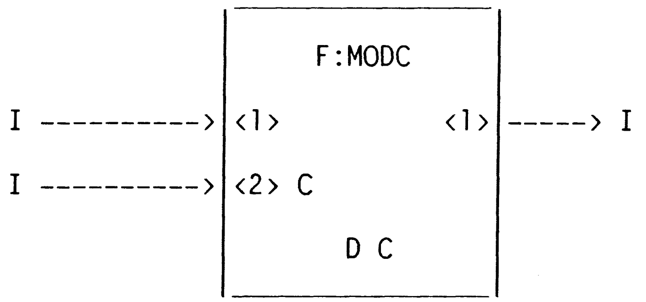
OUTPUT
<1> - remainder from dividing input <1> by input <2>

## NOTES

F:MOD uses a Pascal-like definition of modulo. For a negative integer on input <1>, the resulting output will be negative. For example, -8 mod 3 is -2.

## ASSOCIATED FUNCTIONS

F:MODC



## PURPOSE

Accepts two integers as inputs and produces an integer output that is the remainder resulting from the division of the value on input <1> by the value on input <2>. Input <2> is a constant.

## DESCRIPTION

INPUT
<1> – integer
<2> – integer (constant)

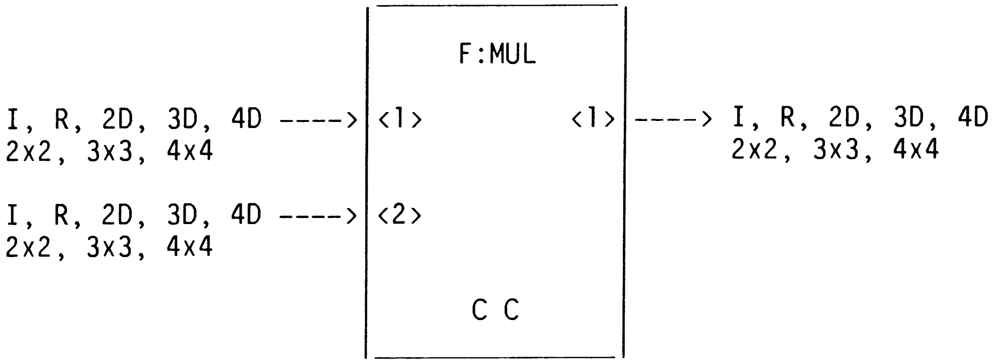
OUTPUT
<1> – remainder from dividing input <1> by input <2>

## NOTES

F:MODC uses a Pascal-like definition of modulo. For a negative integer on input <1>, the resulting output will be negative. For example, –8 mod 3 is –2.

## ASSOCIATED FUNCTIONS

F:MOD


F:MUL
I, R, 2D, 3D, 4D ----> <1>
2x2, 3x3, 4x4
I, R, 2D, 3D, 4D ----> <2>
2x2, 3x3, 4x4
<1> ----> I, R, 2D, 3D, 4D
2x2, 3x3, 4x4
C C

## PURPOSE

Accepts two inputs and outputs their product.

## DESCRIPTION

INPUT
<1> - value
<1> - value

OUTPUT
<1> - product

## NOTES

The two input values must be compatible data types; the output data type depends on the combination of input data types. Vectors are taken to be either row or column vectors, as appropriate, to perform the multiplication.

## ASSOCIATED FUNCTIONS

F:MULC, F:CMUL

```
                          ____________________
                         |                    |
                         |       F:MULC       |
                         |                    |
I, R, 2D, 3D, 4D ---->   |<1>              <1>|----> I, R, 2D, 3D, 4D
2x2, 3x3, 4x4            |                    |      2x2, 3x3, 4x4
                         |                    |
I, R, 2D, 3D, 4D ---->   |<2> C               |
2x2, 3x3, 4x4            |                    |
                         |                    |
                         |        D C         |
                         |____________________|
```

PURPOSE

Accepts two inputs and outputs their product. Input <2> is a constant.

DESCRIPTION

INPUT
<1> - value
<1> - value (constant)

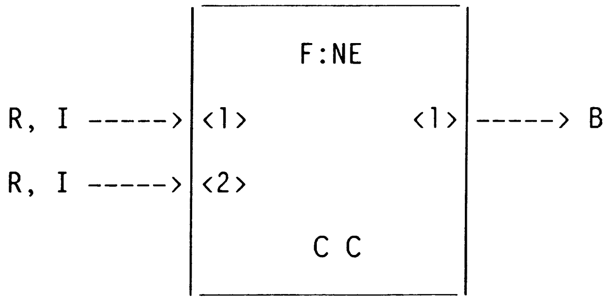
OUTPUT
<1> - product

NOTES

The two input values must be compatible data types; the output data type depends on the combination of input data types. Vectors are taken to be either row or column vectors, as appropriate, to perform the multiplication.

ASSOCIATED FUNCTIONS

F:MUL, F:CMUL



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is not equal to input <2>, and FALSE otherwise.

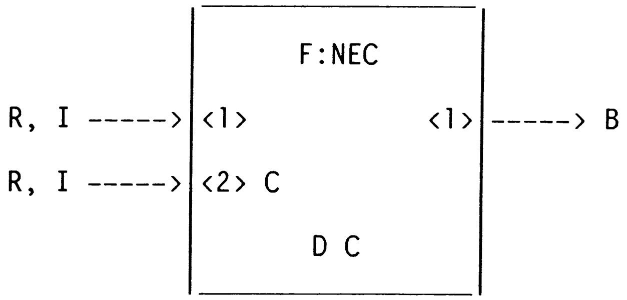
## DESCRIPTION

INPUT
<1> - value to be compared
<2> - value to be compared

OUTPUT
<1> - TRUE if input <1> is not equal to input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:NEC



## PURPOSE

Accepts any combination of reals and integers on its two inputs, and produces a Boolean output that is TRUE if input <1> is not equal to input <2>, and FALSE otherwise. Input <2> is a constant.

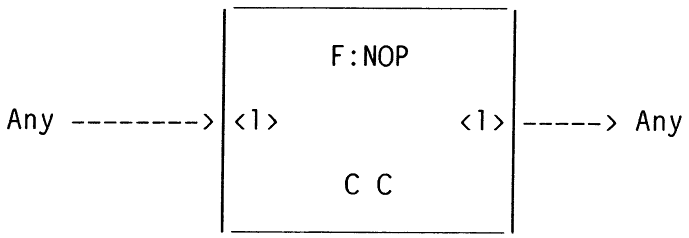
## DESCRIPTION

INPUT
<1> - value to be compared
<2> - value to be compared (constant)

OUTPUT
<1> - TRUE is input <1> is not equal to input <2>, otherwise FALSE

## ASSOCIATED FUNCTIONS

F:NE



## PURPOSE

Accepts any message and outputs that message unchanged.

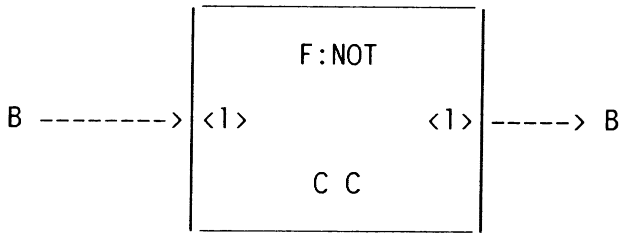
## DESCRIPTION

INPUT
<1> – any message

OUTPUT
<1> – message on input <1>

## NOTES

This function is useful for tying a set of many outputs to a set of many inputs.



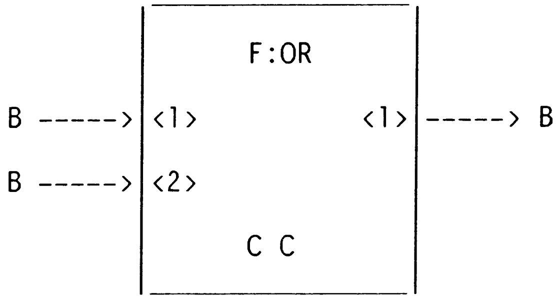
## PURPOSE

Accepts a Boolean input and outputs its complement as a Boolean value.

## DESCRIPTION

INPUT
<1> - Boolean

OUTPUT
<1> - logical complement of input <1>



## PURPOSE

Accepts two Booleans as input and produces a Boolean output that is the logical OR of the two inputs.

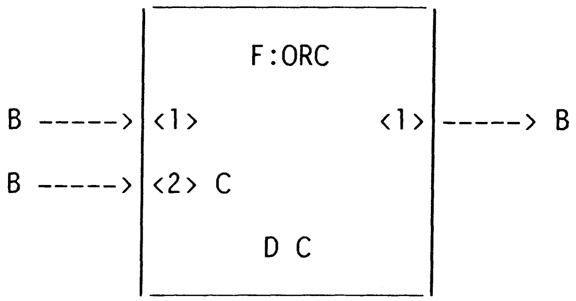
## DESCRIPTION

INPUT
<1> – Boolean
<2> – Boolean

OUTPUT
<1> – logical OR of the two inputs

## ASSOCIATED FUNCTIONS

F:ORC



## PURPOSE

Accepts two Booleans as input and produces a Boolean output that is the logical OR of the two inputs. Input <2> is a constant.

## DESCRIPTION

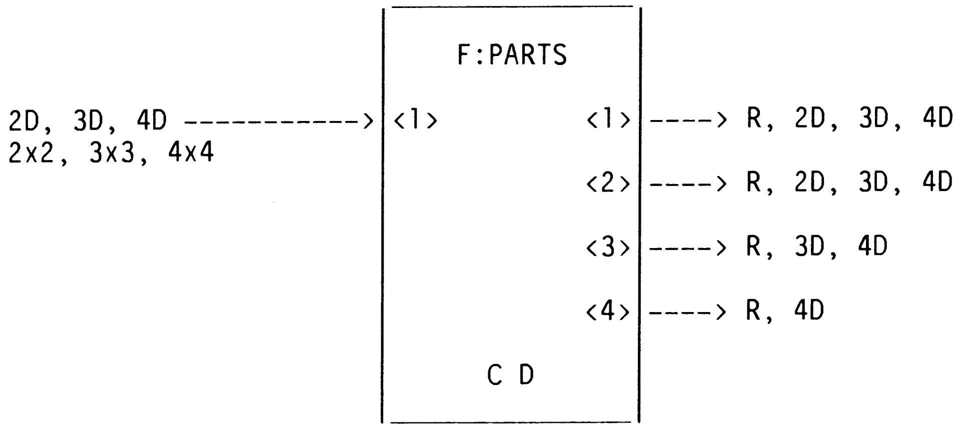
INPUT
<1> - Boolean
<2> - Boolean (constant)

OUTPUT
<1> - logical OR of the two inputs

## ASSOCIATED FUNCTIONS

F:OR

# F:PARTS



```
                                    F:PARTS

2D, 3D, 4D -----------> <1>                   <1> ----> R, 2D, 3D, 4D
2x2, 3x3, 4x4
                                              <2> ----> R, 2D, 3D, 4D

                                              <3> ----> R, 3D, 4D

                                              <4> ----> R, 4D

                                      C D
```

## PURPOSE

Separates a vector into its elements or a square matrix into its row vectors.

## DESCRIPTION

### INPUT

<1> - any vector or matrix

### OUTPUT

<1> - x component or row vector
<2> - y component or row vector
<3> - z component or row vector
<4> - w component or row vector

## NOTES

1. If a square matrix is sent to input <1>, its row vectors appear in sequence at the outputs.

2. If a vector is input, its components are output as real numbers. The x component is output on output <1>, the y component on output <2>, and the z and w components (if any) on output <3> and output <4> respectively.

3. Note that some outputs are not always used. For example, if a 3x3 matrix or a 3D vector is sent to F:PARTS, nothing is output on output <4>.

```
                  _________________
                 |                 |
                 |  F:PASSTHRU(n)  |
                 |                 |
   Any ------->  |<1>          <1> |----> Any
                 | .               |
                 | .               |
                 | .               |
   Any ------->  |<n>          <n> |----> Any
                 |                 |
                 |       D D       |
                 |_________________|
```

PURPOSE

Immediately passes the message which arrives at any input to all function queues connected to its associated output.

DESCRIPTION

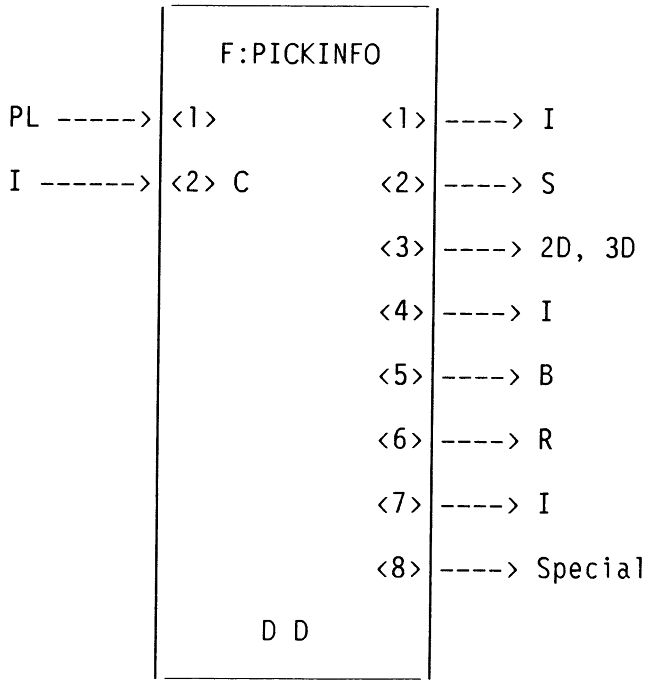
INPUT
<1> – Any message
<n> – Any message

OUTPUT
<1> – Message on input <1>
<n> – Message on input <n>

NOTES

1. A message is passed through as soon as it arrives at an input queue. The function does not have to wait for messages on all its inputs before it becomes active.

2. The **SETUP CNESS** command cannot be used with this function.



## PURPOSE

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

## DESCRIPTION

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

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

## DEFAULTS

The default depth on input <2> is all.

## NOTES

1. Input <1> accepts a picklist. Since the only source of a picklist is the initial function instance PICK, instances of F:PICKINFO must be connected to PICK.

2. Input <2> accepts an integer that specifies the depth within a structure that will be reported when a pick occurs. For example, if the picked item were at the fiftieth level within pick identifiers (i.e., the picked data could be appended with 49 pick identifiers separated by commas) and the integer 2 were input on input <2>, then only the identifier of the picked item and the item directly above it in the structure would be output as the string on output <2>.

3. The output information varies with the type of picklist supplied. If the associated PICK function instance has a TRUE on input <2>, it supplies a detailed coordinate picklist, and most or all of F:PICKINFO's outputs are activated. If the associated PICK has a FALSE on input <2>, a less detailed picklist is supplied, and only F:PICKINFO outputs <1>, <2>, and <5> are activated.

4. The integer on output <1> is the pick index, indicating which vector (in a vector list), character (in a character string), label (in a labels block), or parameter value (in a POLYNOMIAL or RATIONAL POLYNOMIAL curve) was picked. Vectors (or characters or labels) are assigned consecutive integer values in order of their appearance in the list (or string or labels block), beginning with 1.

5. Output <2> is a string containing the requested pick ID's.

6. Output <3> is a 2D or 3D vector giving the coordinates of the intersection of the pickbox with the picked vector. Its data type depends on the data type of the picked vector (2D or 3D). Output <3> also reports the start location of a picked character string or label. (This output is supplied only for coordinate picklists.)

7. Output <4> gives the dimension (2 or 3) of the picked vector. (This output is supplied only for coordinate picklists.)

(continued)

NOTES (continued)

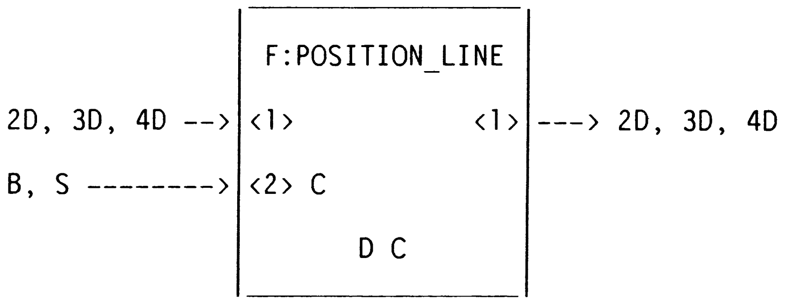
8. Output <5> is TRUE if coordinate picking information is being sent out, and FALSE otherwise. Output <5> is also false if coordinate picking is attempted on a character.

9. Output <6> gives the value of a polynomial parameter t (from 0 to 1, inclusive). This output is activated only for coordinate picklists resulting from picking a vector created by the POLYNOMIAL command or RATIONAL POLYNOMIAL command.

10. Output <7> is for an integer code specifying the data type of the object picked. The code may have values 1 through 8, corresponding to the following data types: (1) CHARACTERS; (2) 2D vector; (3) 3D vector (4) 2D POLYNOMIAL or RATIONAL POLYNOMIAL; (5) 3D POLYNOMIAL or RATIONAL POLYNOMIAL; (6) 2D BSPLINE or RATIONAL BSPLINE; (7) 3D BSPLINE or RATIONAL BSPLINE; (8) LABELS.

11. When output <8> is connected to <1>F:PRINT it causes F:PRINT to produce the name of the VECTOR_LIST, CHARACTERS, LABELS, BSPLINE, RATIONAL BSPLINE, POLYNOMIAL, or RATIONAL POLYNOMIAL command containing the picked vector.

12. If the command containing the picked vector is not named, a null is output at <8>.



## PURPOSE

Accepts a 2D, 3D, or 4D vector on input <1>. A Boolean on input <2> is used to assign a position (P) or line (L) to be associated with the vector. A string sent to input <2> consists of either a P or an L identifier. The vector, with the position/line condition specified by the Boolean, is output on output <1>.

## DESCRIPTION

INPUT
<1> - any vector
<2> - Boolean or string (constant)

OUTPUT
<1> - vector with P or L identifier

## NOTES

A TRUE on input <2> causes a line (L) to be associated with the vector; a FALSE on input <2> causes a position (P) to be associated with the vector. The outputs from this function (vectors with position/line specifications) can only be applied to a vector list data node in a display tree. No function accepts such vectors as inputs.

```
                 ________________
                |                |
                |     F:PRINT    |
                |                |
Any ----------->|<1>          <1>|-------> S
                |                |
B ------------->|<2> C           |
                |                |
                |       D C      |
                |________________|
```

## PURPOSE

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

## DESCRIPTION

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

OUTPUT
<1> - string

## DEFAULTS

The default for input <2> is FALSE, indicating decimal format.

## NOTES

1. Any message on input <1> is converted to string format and sent out on output <1>.

NOTES (continued)

2. Input <2> governs the format of real numbers and vectors (but not matrix elements) in the output string. When input <2> is FALSE, these values have the usual decimal format (e.g., '.001'). When input <2> is TRUE, these values are in exponential format (e.g. '1.000000E-3'). (Integers, on the other hand, are never in exponential format.) The output character string that results from each type of input follows:

| Input Data Type | Output Character String |
|---|---|
| Boolean | 'FALSE' or 'TRUE'. |
| Character | The same character that was input. |
| String | The same character string that was input. |
| Integer | The character representation of the integer; e.g., '129', '-107543'. |
| Real | A character representation of the real number; e.g., '3.1416', '2.3E2' etc. |

All vectors are preceded by a P (position), L (line), or V (no P or L) designation. ("X" in the following vector descriptions indicates P, L, or V.)

| Input Data Type | Output Character String |
|---|---|
| 2D Vector | Two real numbers separated by a comma; e.g., 'X 3.5,.0715' |
| 3D Vector | Three real numbers separated by commas; e.g., 'X 3.1416,-275.012,3.5' |
| 4D Vector | Four real numbers separated by commas; e.g., 'X 3.1416,-275.012,3.5,.0715' |

NOTES (continued)

Input Data Type — Output Character String

2x2 Matrix

Two 2D vectors (nine-digit precision, exponential format) separated by a space; e.g., '1.23456789E01, -2.56900187E-02 3.14159265E01, 2.71828183E01')

3x3 Matrix

Three 3D vectors (nine-digit precision, exponential format) separated by spaces.

4x4 Matrix

Four 4D vectors (nine-digit precision, exponential format) separated by spaces.

Pick list

The format of a picklist string depends on whether coordinate information was requested for the picklist (refer to F:PICKINFO and the PICK initial function instance) and, if it was requested, whether it was given. (For example, a vector in a character is not susceptible to standard coordinate picking.) All of these formats contain the clause <pick ID's>. This clause contains two things: first, a list of pick identifiers established in SET PICK ID, with the "closest" pick identifier first; second, a space followed by the name of the original data-definition command corresponding to the picked object. If this command is not named, neither a name nor a space follows the pick identifiers.

If no coordinate picking information was requested (input <2> of the associated PICK function instance is FALSE), the output string has the format

```
<index><pick ID's>
```

for a vector in a declared vector list (including WITH PATTERN lists) or for a character in a string or label in a block, and

```
< ><pick ID's>
```

for a vector in a polynomial curve.

NOTES (continued)

Input Data Type — Output Character String

Pick list (cont.)

If coordinate picking information was requested and given (i.e., if input <2> of the associated PICK is TRUE, and it was not a character vector), then the output string format is

```
<1><dimension><pick_x, pick_y, [pick_z]>
<index><pick ID's>
```

for a vector in a declared vector list and

```
<2><dimension><pick_x, pick_y,[pick_z]><t>
<pick ID's>
```

for a vector within a polynomial curve, where <dimension> and <t> are as defined for F:PICKINFO.

For a character in a string the format is

```
<3><dimension><start_x, start_y, start_z>
<index><pick ID's>
```

and for a label in a labels block, the format is

```
<5><dimension><start_x, start_y, start_z>
<index><pick ID's>
```

If picked coordinates were requested but not given (i.e., input <2> of the associated PICK is TRUE and a vector in a character or in a polynomial curve was picked), the output string format is

```
<3><index><pick ID's>
```

# F:PUT STRING

```
                 ____________________
                |                    |
                |    F:PUT_STRING    |
  S ------>     | <1>            <1> | ----> S
  I ------>     | <2>            <2> | ----> B
  S ------>     | <3>                |
                |        C C         |
                |____________________|
```

## PURPOSE

Replaces characters in the string on input <1> with the string on input <3>, starting at the position specified by the integer on input <2>. The resulting string may be longer than the original string if the string on input <3> overlaps. The Boolean on output <2> is TRUE if the resulting string is the same length as the string on input <1>, and FALSE otherwise.

## DESCRIPTION

INPUT

<1> - string
<2> - starting location for replacing characters
<3> - replacement characters

OUTPUT

<1> - resulting string
<2> - TRUE = resulting string same length as the original, FALSE = resulting string longer than the original

```
                          _______________________
                         |                       |
                         |    F:RANGE_SELECT     |
                         |                       |
R, 2D, 3D ---------->    |<1>                 <1>|-----> R, 2D, 3D
                         |                       |
R, 2D, 3D ---------->    |<2> C               <2>|-----> R, 2D, 3D
                         |                       |
R, 2D, 3D ---------->    |<3> C               <3>|-----> R, 2D, 3D
                         |                       |
                         |          D D          |
                         |_______________________|
```

## PURPOSE

Compares the value on input <1> to the maximum and minimum on inputs <2> and <3> to determine whether the value is in range or not.

## DESCRIPTION

INPUT
<1> - value
<2> - maximum (constant)
<3> - minimum (constant)

OUTPUT
<1> - in-range, normalized
<2> - in-range, unchanged
<3> - out-of-range, unchanged

## NOTES

1. Accepts real number values or 2D or 3D vectors on all inputs. The data type must be the same on all inputs, as must the vector dimensions (that is, all vectors must be either 2D or 3D). The type of data output from the function is the same type that is input to the function.

2. The value on input <1> is compared to the constant maximum value on input <2> and the constant minimum value on input <3>.

NOTES (continued)

3. If the value on input <1> is within the range defined by the minimum and maximum values (input <3> <= input <1> <= input <2>) then the value on input <1> is sent out on outputs <1> and <2>..

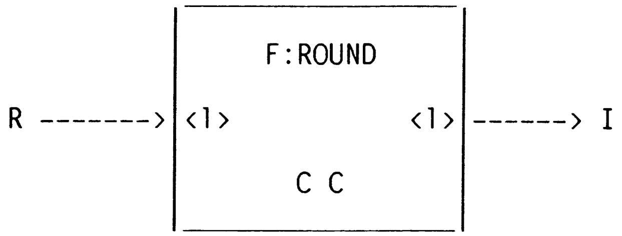
4. The value on output <1> is normalized to the maximum/minimum values of inputs <2> and <3>. The value on output <2> is identical to the input <1> value. If the value is in range, nothing is sent out on output <3>.

5. If the value on input <1> is not within range, it is output on output <3> unchanged. Data is normalized for output <1> by:

$$\text{normal X value} = \frac{X^1 - X^{min}}{\text{X range}} - 0.5$$

$$\text{normal Y value} = \frac{Y^1 - Y^{min}}{\text{Y range}} - 0.5$$

$$\text{normal Z value} = \frac{Z^1 - Z^{min}}{\text{Z range}} - 0.5$$



## PURPOSE

Accepts a real number and outputs the nearest integral value.

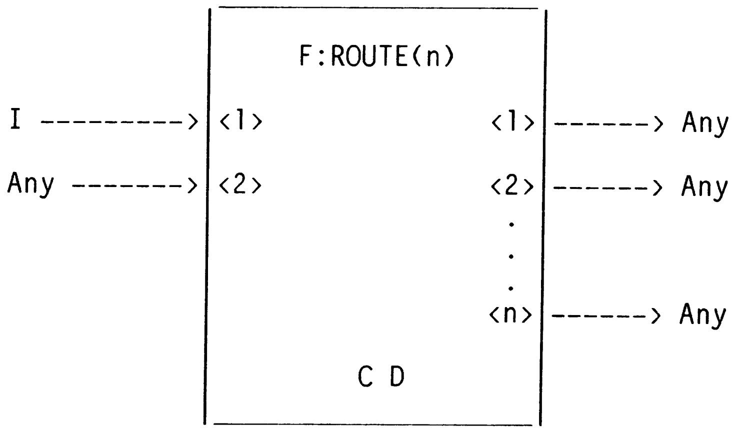
## DESCRIPTION

INPUT
<1> - real number

OUTPUT
<1> - nearest integral value

## NOTES

Values n to n.4999...9 are rounded to n; values n.5 to n.9999...9 are rounded to n+1. Values -n to -n.4999...9 are rounded to -n; values -n.5 to -n.999...9 are rounded to -n+(-1).



## PURPOSE

Uses the integer on input <1> to route the message on input <2> to the output whose number matches the input <1> integer.

## DESCRIPTION

INPUT
<1> - number of selected output (1 through n)
<2> - any message

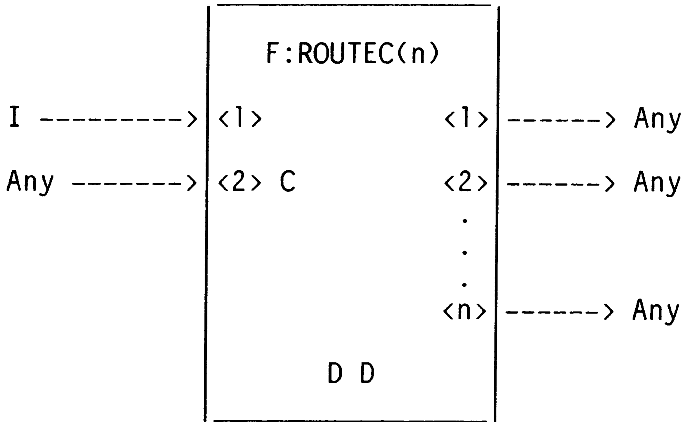
OUTPUT
<1> - message on input <2>
.
.
<n> - message on input <2>

## NOTES

The message on input <2> may be of any data type. The "n" in the function name can be any integer from 2 to 127. If the integer on input <1> is not a number from 1 to n inclusive, then an error is detected and reported.

## ASSOCIATED FUNCTIONS

F:ROUTEC(n) and F:CROUTE(n)



## PURPOSE

Uses the integer on input <1> to switch the message on input <2> to the output whose number matches the input <1> integer. Input <2> is a constant.

## DESCRIPTION

INPUT
<1> – number of selected output (1 through n)
<2> – any message (constant)

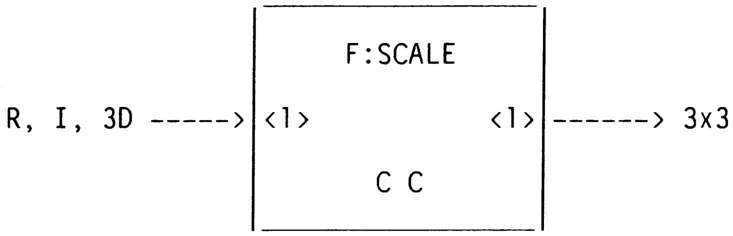
OUTPUT
<1> – message on input <2>
.
.
<n> – message on input <2>

## NOTES

The message on input <2> may be of any data type. The "n" in the function name may be any integer from 2 to 127. If the integer on input <1> is not a number from 1 to n, inclusive, then the message on input <2> is held until a valid integer is received on input <1>.

## ASSOCIATED FUNCTIONS

F:ROUTE(n), F:CROUTE(n)



## PURPOSE

Accepts a real value, an integer, or a 3D vector. If a real is input, the scaling factor represented by the real value is applied to X, Y, and Z. A 3x3 scaling matrix is output that may be used to update a scaling element of a display data structure.

## DESCRIPTION

### INPUT

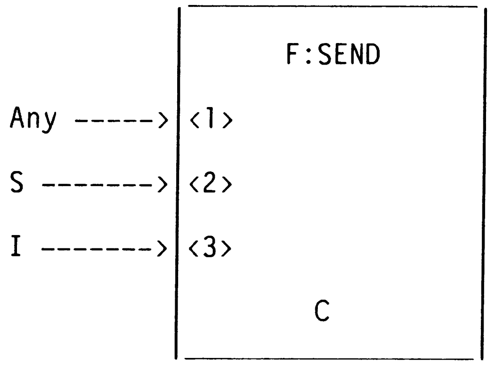
<1> - value

### OUTPUT

<1> - 3x3 scaling matrix

## NOTES

If a 3D vector is input, the X component of the vector is the scaling factor for X, the Y component of the vector is the scaling factor for Y and the Z component of the the vector is the scaling factor for Z.



## PURPOSE

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

## DESCRIPTION

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

## NOTES

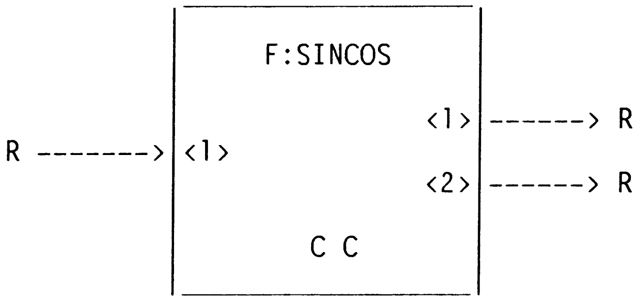
1. This function has no output.

2. Input <1> accepts special data types that most functions do not accept, such as the data type output by F:LABEL.

3. The SETUP CNESS command can be used to specify constant inputs as default values.

# F:SINCOS

Intrinsic Function
Arithmetic and Logical



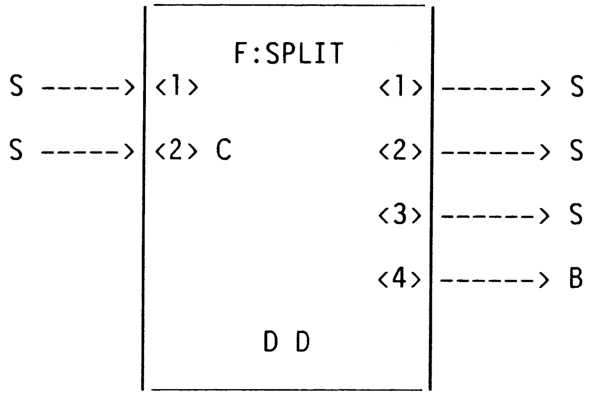
## PURPOSE

Accepts a real number on input <1> which represents an angle in units of degrees. The sine of that angle is output as a real number on output <1>, and the cosine of that angle is output as a real number on output <2>.

## DESCRIPTION

INPUT
<1> - angle

OUTPUT
<1> - sine
<2> - cosine



## PURPOSE

Accepts character strings on inputs <1> and <2>. The string on input <2> is a constant. When the string is received on input <1>, it is compared to the string on input <2> for an exact match.

## DESCRIPTION

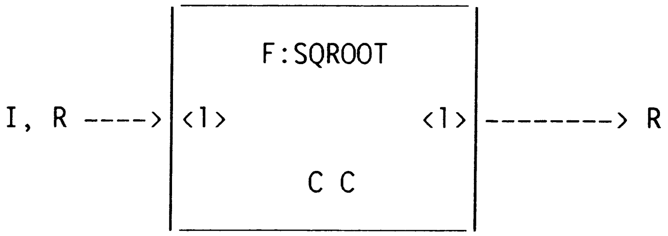
INPUT
- <1> - string
- <2> - string (constant)

OUTPUT
- <1> - characters preceding match
- <2> - matching characters
- <3> - characters following match
- <4> - TRUE if matching inputs, FALSE otherwise

## NOTES

1. If a match occurs, characters in the string on input <1> that precede the match are output on output <1>. Matching characters are output on output <2>. Characters following the matching characters are output on output <3>. And a Boolean TRUE is output on output <4>.

2. If no match is found, nothing is output on outputs <1>, <2>, and <3>, and a Boolean FALSE is output on output <4>.



PURPOSE

Extracts the square root of the real number or integer on input <1>.

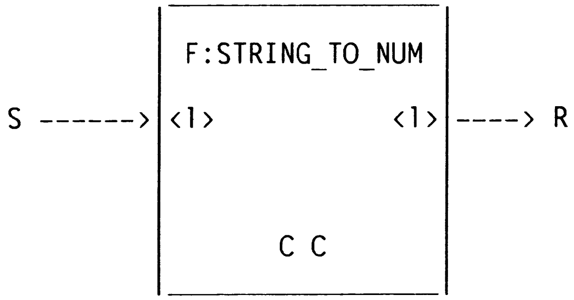
DESCRIPTION

INPUT
<1> - real or integer

OUTPUT
<1> - square root

NOTES

The output is always real. If the input is negative, the output is 0.



## PURPOSE

Outputs the value of a string of digits as a real number. If the function receives characters that cannot represent a number then an error message is generated.

## DESCRIPTION

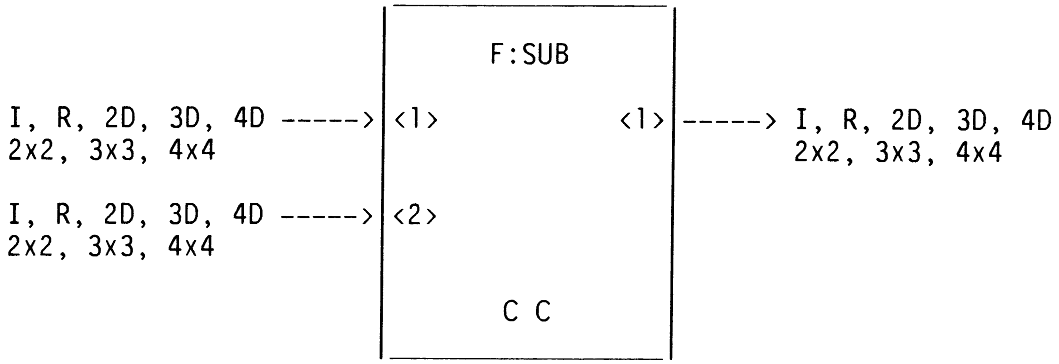
INPUT
<1> - string of digits

OUTPUT
<1> - value of the string on input <1>

## NOTES

A valid number can contain any or all of the following components: decimal point, 'E' expression, plus or minus sign, numerals.

# F:SUB



## PURPOSE

Accepts two inputs and produces an output that is the difference of the two inputs (input <2> is subtracted from input <1>).

## DESCRIPTION

INPUT
<1> - minuend
<2> - subtrahend

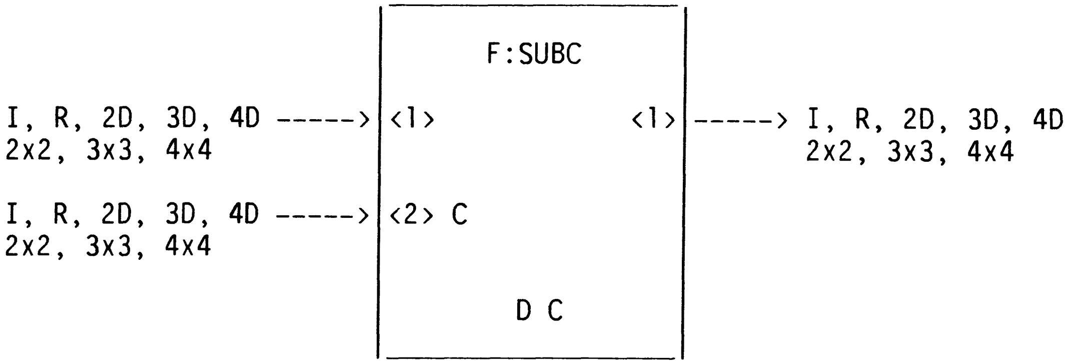
OUTPUT
<1> - difference

## NOTES

The two input values must be of the same data type (except a combination of real and integer is allowed); the output data type depends on the input data type. If a real and an integer are input, a real is output.

## ASSOCIATED FUNCTIONS

F:SUBC, F:CSUB



## PURPOSE

Accepts two inputs and produces an output that is the difference of the two inputs (input <2> is subtracted from input <1>). Input <2> is a constant.

## DESCRIPTION

INPUT
<1> – minuend
<2> – subtrahend (constant)

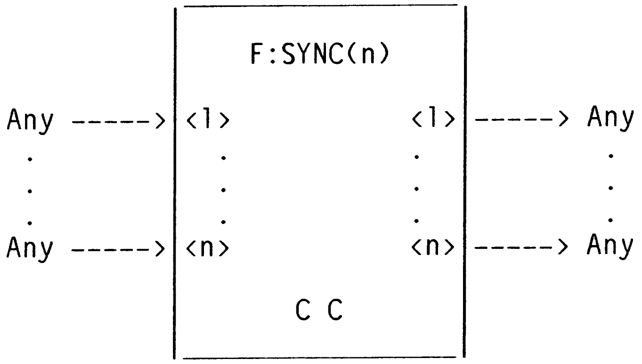
OUTPUT
<1> – difference

## NOTES

The two input values must be of the same data type (except a combination of real and integer is allowed); the output data type depends on the input data type. If a real and an integer are input, a real is output.

## ASSOCIATED FUNCTIONS

F:SUB, F:CSUB



## PURPOSE

Synchronizes the output of a specified number of messages. The number, "n", may have any value from 2 to 127.

## DESCRIPTION

INPUT

<1> - any message
.
.
<n> - any message

OUTPUT

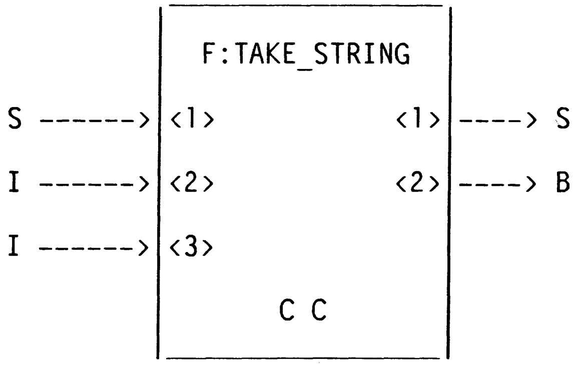
<1> - any message
.
.
<n> - any message

## NOTES

1. F:SYNC(n) waits until a message is received on all of its "n" inputs, then sends the messages out; for example, F:SYNC(32) synchronizes 32 messages.

2. Usually, the outputs of an F:SYNC(n) function instance are connected to nodes in a display tree to assure that updates to displayed data are synchronized.

3. Outputs from F:SYNC(n) are effectively simultaneous. In fact, outputs are sequential (<1> through <n>) at a rapid rate.



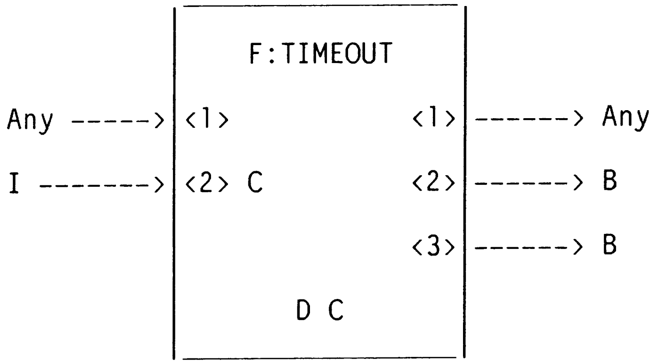
## PURPOSE

Outputs a string consisting of the number of characters specified on input <2> taken from the string on input <1>, starting at the position given on input <3>. A TRUE on output <2> means that there were enough characters left in the string. A FALSE means there were not enough characters, so the output string was truncated.

## DESCRIPTION

INPUT
<1> - string
<2> - starting position
<3> - number of characters to take

OUTPUT
<1> - resulting string
<2> - TRUE = enough characters, FALSE = output string truncated



## PURPOSE

Provides the means to detect the occurrence of consecutive messages on input <1> within the time interval specified in centiseconds by the constant integer on input <2>.

## DESCRIPTION

INPUT
<1> - message on input <1>
<2> - time interval (constant)

OUTPUT
<1> - any message
<2> - TRUE = timeout, FALSE = no timeout
<3> - logical complement of output <2>

## NOTES

1. Once the first message is received on input <1>, the subsequent message must be received in the duration specified on input <2> in order to be passed through the function. Then the third message must be received within that specified duration after the second message, and so on.

2. The first message to input <1> serves only to start the timeout measurement, and never generates an output.

NOTES (continued)

3. If any subsequent messages are received at input <1> within the time interval specified on input <2>, only the last message is sent on output <1> at the end of the interval; all intervening messages are discarded.

4. If a message on input <1> is not received within the specified time, the Boolean on output <2> is TRUE. If a message on input <1> is received within the interval, the Boolean on output <2> is FALSE. Output <3> is the complement of output <2>.

5. This function is especially useful to determine a data tablet stylus out-of-range condition. If the message from the data tablet stylus is connected to input <1> of this function and an appropriate duration is specified on input <2>, then the inputs from the data tablet will be passed through the function until the duration is exceeded.

```
            |                  |
            |  F:TRANS_STRING  |
S ------>   |<1>            <1>|----> S
I ------>   |<2> C             |
S ------>   |<3> C             |
            |       D C        |
            |__________________|
```

## PURPOSE

Translates the string on input <1> into the output string using the string on input <3> as a translation table. The integer on input <2> is the beginning place (i.e., the ASCII decimal equivalent or ORD) of the first character to be translated. Inputs <2> and <3> are constants.

## DESCRIPTION

INPUT
<1> - string
<2> - first character to be translated (constant)
<3> - translation table (constant)

OUTPUT
<1> - translated string

## NOTES

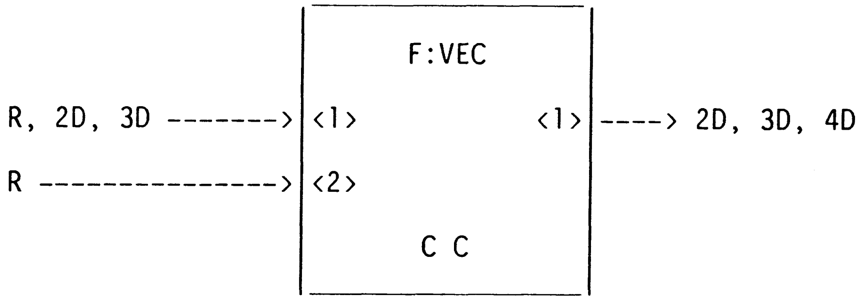
1. The upper-limit of the number of characters to translate is the length of the string on input <3>

## EXAMPLE

```
SEND 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' TO <3>Trans_String;
SEND FIX(97) TO <2>Trans_String;      { the ASCII equivalent of 'a'}
SEND 'abcdefghijklmnopqrstuvwxyz' TO <1>Trans_String;
```

The lower case letters send to input <1> will be translated to upper case on output <1>.

# F:VEC



## PURPOSE

Accepts two real numbers and outputs a 2D vector, accepts a 2D vector and a real number and outputs a 3D vector, or accepts a 3D vector and a real number and outputs a 4D vector.

## DESCRIPTION

### INPUT

<1> – real, 2D, or 3D vector
<2> – real number

### OUTPUT

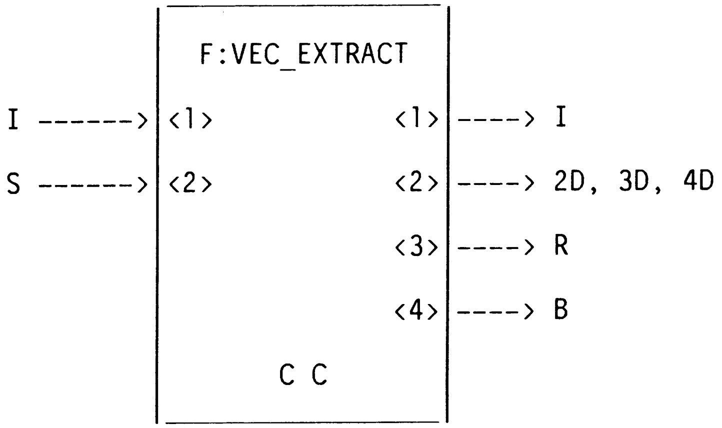
<1> – vector consisting of the value on input <1> with the real on
input <2> appended

## NOTES

The output vector is the real number or vector from input <1> with the real number from input <2> appended as the last vector component.

## ASSOCIATED FUNCTIONS

F:VECC, F:CVEC



## PURPOSE

Extracts information about a vector in a vector list node given an index into the vector list on input <1> and the name of the vector list node on input <2>.

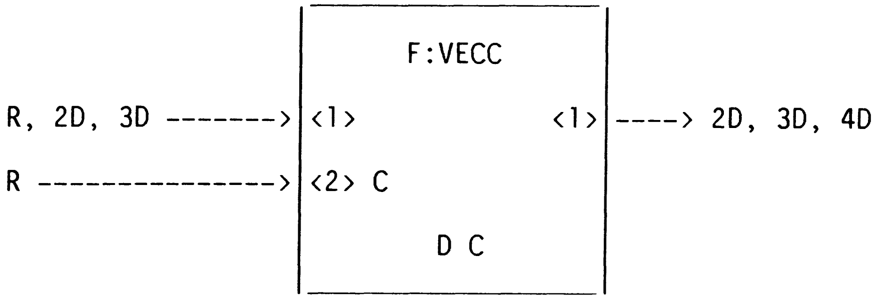
## DESCRIPTION

INPUT
<1> – index of the vector in question
<2> – name of the vector list node

OUTPUT
<1> – data type
<2> – the vector in question
<3> – intensity
<4> – TRUE = Position, FALSE = Line

## NOTES

1. The integer on output <1> is the same as would be sent from output <7> of F:PICKINFO.



## PURPOSE

Accepts two real numbers and outputs a 2D vector, accepts a 2D vector and a real number and outputs a 3D vector, or accepts a 3D vector and a real number and outputs a 4D vector. Input <2> is a constant.

## DESCRIPTION

INPUT
<1> - real, 2D, or 3D vector
<2> - real number (constant)

OUTPUT
<1> - vector consisting of the value on input <1> with the real on input <2> appended

## NOTES

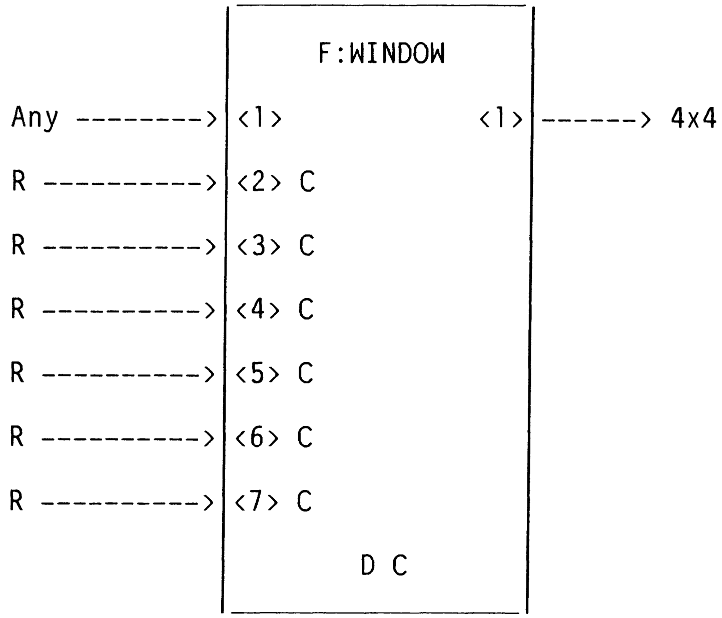
The output vector is the real number or vector from input <1> with the real number from input <2> appended as the last vector component.

## ASSOCIATED FUNCTIONS

F:VEC, F:CVEC

# F:WINDOW

Intrinsic Function
Viewing Transformation



## PURPOSE

This is the functional counterpart of the **WINDOW** command. The windowing matrix that results from this function defines a viewing area for orthographic views (parallel projections) of objects.

## DESCRIPTION

INPUT
<1> - trigger
<2> - x minimum (constant)
<3> - x maximum (constant)
<4> - y minimum (constant)
<5> - y maximum (constant)
<6> - z minimum (constant)
<7> - z maximum (constant)

OUTPUT
<1> - 4x4 matrix

NOTES

1. **F:WINDOW** accepts any message on input <1> to trigger the function and constant real values on inputs <2> through <7>. These real values define the boundaries of a three-dimensional rectangular volume within which objects can be viewed in parallel projection (i.e. no perspective is imposed).

2. This volume is defined by expressing a rectangle in terms of **xmin** (input <2>), **xmax** (input <3>), **ymin** (input <4>), and **ymax** (input <5>). The rectangle is then extended into a three dimensional volume by specifying **zmin** (input <5>) and **zmax** (input <7>).

# F:XFORMDATA

Intrinsic Function
Data Conversion

```
                 ____________________
                |                    |
                |    F:XFORMDATA     |
Any --------->  | <1>           <1>  | ------> Special
S ----------->  | <2> C              |
S ----------->  | <3> C              |
I ----------->  | <4> C              |
I ----------->  | <5> C              |
                |        D C         |
                |____________________|
```

## PURPOSE

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

## DESCRIPTION

INPUT

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

OUTPUT

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

## DEFAULTS

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

NOTES

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

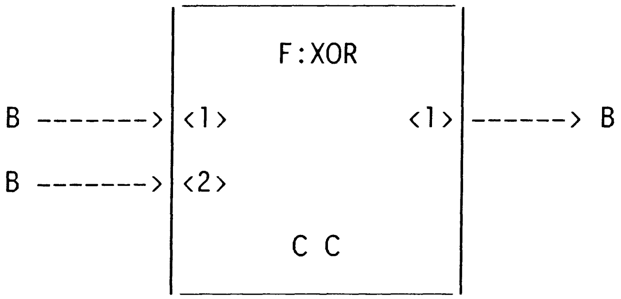
2. Input <2> is a string or matrix containing the name of the XFORM command in the display tree (either XFORM MATRIX or XFORM VECtor). By referring to an XFORM command, this input indirectly specifies the object whose transformed data is to be sent. If the string names something other than an XFORM command,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

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

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

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

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



## PURPOSE

Accepts Boolean values on inputs <1> and <2>, performs an exclusive-OR function on the values, and outputs the result as a Boolean value. That is, if the Boolean values on both inputs are the same, the output is FALSE; if the Boolean values on the inputs are different, the output is TRUE.

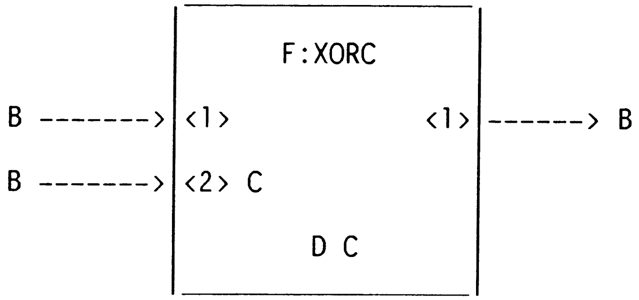
## DESCRIPTION

INPUT
<1> - Boolean
<2> - Boolean

OUTPUT
<1> - exclusive OR of inputs

## ASSOCIATED FUNCTIONS

F:XORC



## PURPOSE

Accepts Boolean values on inputs <1> and <2>, performs an exclusive-OR function on the values, and outputs the result as a Boolean value. Unlike **F:XOR**, for which both inputs are active, **F:XORC** input <2> is a constant. If the Boolean values on both inputs are the same, the output is FALSE; if the Boolean values on the inputs are different, the output is TRUE.

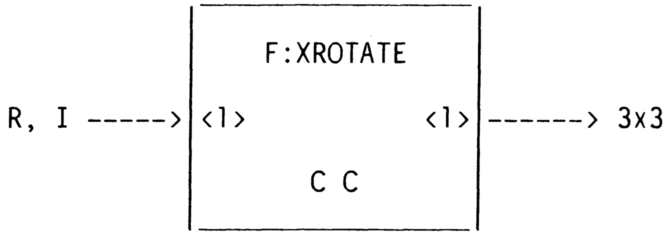
## DESCRIPTION

INPUT
<1> – Boolean
<2> – Boolean (constant)

OUTPUT
<1> – exclusive OR of inputs

## ASSOCIATED FUNCTIONS

F:XOR



PURPOSE

Accepts a real value or an integer that specifies the number of degrees about the X axis that the rotation matrix generated by the function is to represent.

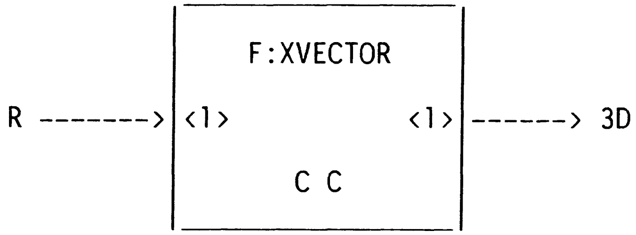
DESCRIPTION

INPUT
<1> - real value or integer

OUTPUT
<1> - 3x3 rotation matrix

NOTES

The 3x3 rotation matrix which is output may be used to update a rotation node in a display tree.



PURPOSE

Accepts a real on input <1> and outputs a 3D vector.

DESCRIPTION

INPUT
<1> - real number

OUTPUT
<1> - 3D vector

NOTES

In the 3D vector which is output, x is equal to the input real, and y and z are 0. For example, if 3 were input, the 3D vector output would be 3,0,0.

```
                 _______________________
                |                       |
                |       F:YROTATE       |
                |                       |
    R, I ------>|<1>                 <1>|-------> 3x3
                |                       |
                |          C C          |
                |_______________________|
```

## PURPOSE

Accepts a real value or an integer that specifies the number of degrees about the Y axis that the rotation matrix generated by the function is to represent.

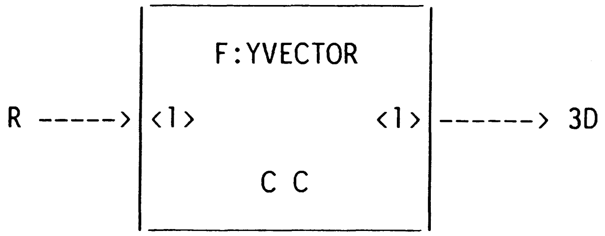
## DESCRIPTION

INPUT
<1> - degrees rotation in Y

OUTPUT
<1> - 3x3 rotation matrix

## NOTES

The 3x3 rotation matrix that is output may be used to update a rotation node in a display tree.



PURPOSE

Accepts a real on input <1> and outputs a 3D vector.

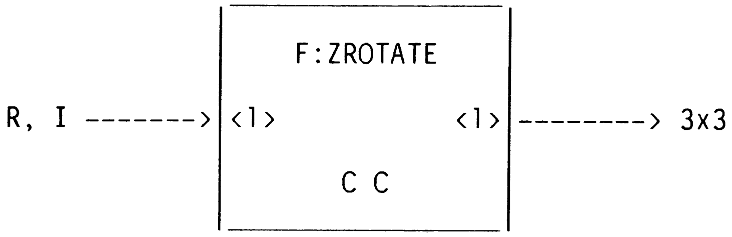
DESCRIPTION

INPUT
<1> - real number

OUTPUT
<1> - 3D vector

NOTES

In the 3D vector which is output, y is equal to the input real, and x and z are 0. For example, if 4 were input, the 3D vector output would be 0,4,0.



## PURPOSE

Accepts a real value or an integer that specifies the number of degrees about the Z axis that the rotation matrix generated by the function is to represent.

## DESCRIPTION

INPUT
<1> - degrees rotation in Z

OUTPUT
<1> - 3x3 rotation matrix

## NOTES

The 3x3 rotation matrix that is output may be used to update a rotation node in a display tree.

```
             ___________________
            |                   |
            |     F:ZVECTOR     |
            |                   |
R ----->    |<1>             <1>|------> 3D
            |                   |
            |        C C        |
            |___________________|
```

PURPOSE

Accepts a real on input <1> and outputs a 3D vector.

DESCRIPTION

INPUT
<1> - real number

OUTPUT
<1> - 3D vector

NOTES

In the 3D vector which is output, z is equal to the input real, and x and y are 0. For example, if 5 were input, the 3D vector output would be 0,0,5.

```
                              BUTTONSIN
                             (BUTTONSIN2)
                        _______________________
                       |                       |
(Connected to---->     | <1>               <1> | ------> I
Function Buttons       |                       |
Unit at system         |                       |
initialization)        |                   <2> | ------> B
                       |                       |
                       |          C C          |
                       |_______________________|
```

## PURPOSE

Detects activity from the Function Buttons Unit on input <1>, which is system-connected at initialization.

## DESCRIPTION

INPUT
<1> - connected to Function Buttons

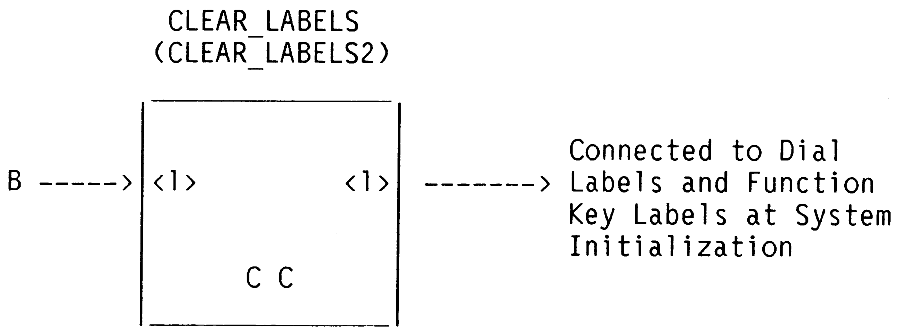
OUTPUT
<1> - number of the button activated
<2> - TRUE = on, FALSE = off

## NOTES

Output occurs when one of the 32 buttons is pushed. The number of the pushed button appears at output <1>, and its light state (TRUE for on, FALSE for off) at output <2>.

## CLEAR_LABELS

Initial Function Instance
Output


CLEAR_LABELS
(CLEAR_LABELS2)
B ----->
<1>
<1>
-------->
Connected to Dial Labels and Function Key Labels at System Initialization
C C

### PURPOSE

Clears the control dial LED labels and the function key LED labels. If input <1> is TRUE, the labels are cleared; otherwise, no action is taken.

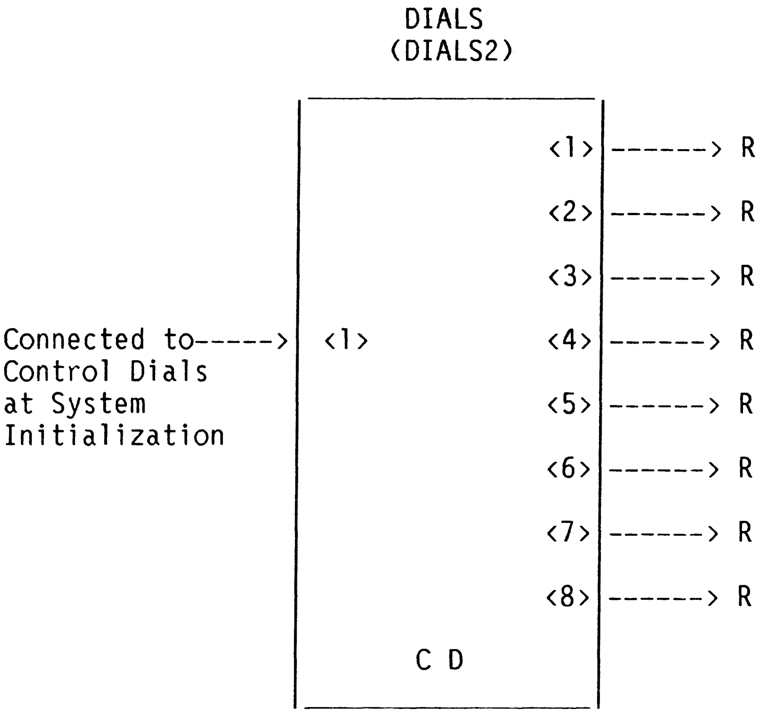
### DESCRIPTION

INPUT
<1> – TRUE = clear labels, FALSE = no action

OUTPUT
<1> – connected to Dial and Function Key labels

### NOTES

The **INITialize** command sends a TRUE to this function instance, clearing all LED labels.



## PURPOSE

Produces eight real number outputs that correspond to inputs from control dials 1 though 8.

## DESCRIPTION

INPUT
<1> - Connected to Control Dials

OUTPUT
<1> - real number
<2> - real number
<3> - real number
<4> - real number
<5> - real number
<6> - real number
<7> - real number
<8> - real number

(continued)

NOTES

1. The control dials are numbered from 1 through 4, left to right across the top row and from 5 to 8, left to right across the bottom row.

2. The message from each control dial is converted to a real number value, which is the incremented value from the dial normalized to between -1.0 and +1.0. This value is sent out on the output (<1>...<8>) that corresponds to the number of the dial that sent the message.

```
                      DLABEL1 ... DLABEL8
                     (DLABEL21 ... DLABEL28)
                    ___________________________
                   |                           |
    S ---------->  |<1>                        |
                   |                        <1>|-----> Connected to
                   |                           |       Dial Labels
    B ---------->  |<2> C                      |       at System
                   |                           |       Initialization
                   |                           |
    B ---------->  |<3> C                      |
                   |                           |
                   |           D C             |
                   |___________________________|
```

## PURPOSE

Eight function instances are provided to separately label the eight LED indicators above each control dial. **DLABEL1** is used to label the LED indicators associated with the first control dial (leftmost, top row); **DLABEL2** is used to label the LED indicators associated with the second control dial (second from left, top row); and so on, through **DLABEL8**, which is used to label the LED indicators associated with the eighth dial (rightmost, bottom row).

## DESCRIPTION

INPUT

<1> – label message
<2> – blink/no blink (constant)
<3> – center/left justify (constant)

OUTPUT

<1> – connected to Control Dials

(continued)

NOTES

1. Input <1> accepts the character string (up to eight characters) to be displayed on the corresponding control dial LED indicators. The constant Boolean on input <2> selects blink (TRUE) or no blink (FALSE) for the displayed characters. The constant Boolean on input <3> controls whether the displayed message will be centered in the eight available locations (TRUE) or whether the first character will be placed in the leftmost of the eight locations (left justified) (FALSE).

2. If inputs <2> and <3> are not used, the message will not blink and will be centered.

3. Allowable characters for control dial labels are:

   ! " # $ % & ' ( ) * + - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z < [ ] _

4. Lowercase letters are converted to uppercase letters. A space may also be specified.

5. Carriage return <CR> and line feed <LF> are not legal characters and cause a message that follows <CR> and/or <LF> to be partially lost or ignored.

```
                     DSET1 ... DSET8
                    (DSET21 ... DSET28)
                  _____________________
                 |                     |
  R, I -------->|<1>               <1>|----> Connected to
                 |                     |      Control Dials
                 |                     |      at System
                 |                     |      Initialization
                 |         C C         |
                 |_____________________|
```

## PURPOSE

Eight function instances are provided to set the operating parameters for the eight control dials. DSET1 is used to set parameters for the first control dial (leftmost, top row); DSET2 is used to set parameters for the second control dial (second from left, top row); and so on, through DSET8, which is used to set parameters for the eighth control dial (rightmost, bottom row).

## DESCRIPTION

INPUT
<1> - real = delta value, integer = sample rate

OUTPUT
<1> - connected to Control Dials

## DEFAULTS

All control dials default to an enabled condition in relative mode (each value from a dial reflects the amount of change (delta) from the last output value). There is no absolute mode for the control dials.

The default sample rate is 20 per second.

(continued)

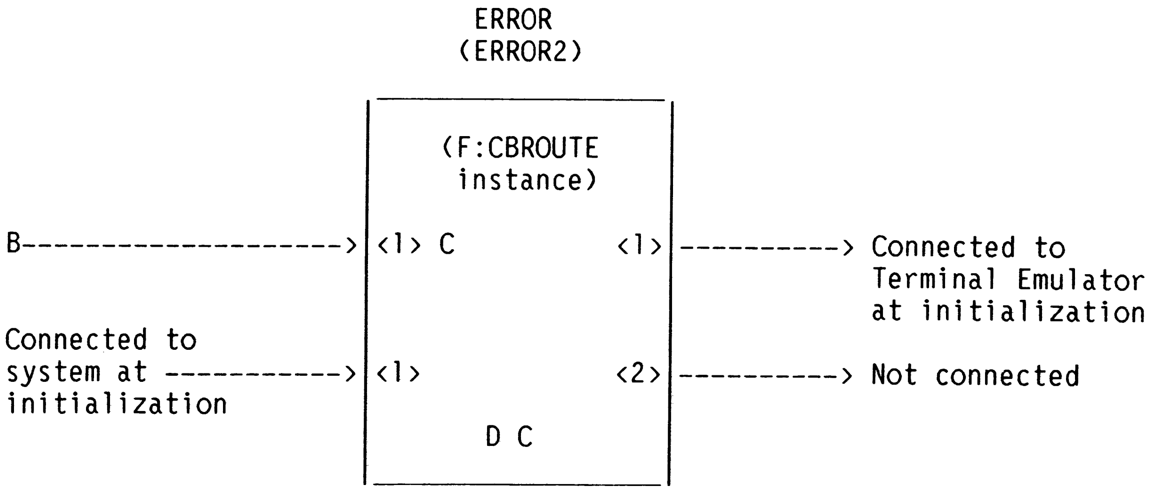
NOTES

1. Input <1> accepts real numbers or integers that set the delta value and sample rate. The default sample rate is 20 samples per second.

2. Real numbers set the delta value relative to one complete dial rotation. For example, if .25 were the real number input, the dial would have to be rotated 90 degrees (.25 x 360) before an output from the dial would be generated.

3. An integer is applied to input <1> to indicate the sample rate. Sample rate is specified in samples per second. For example, the integer 10 causes the dial to be sampled 10 times per second.

4. Output <1> is used to set the dial parameters as specified by the real number or integer on input <1>.



PURPOSE

Enables and disables the display of error messages.

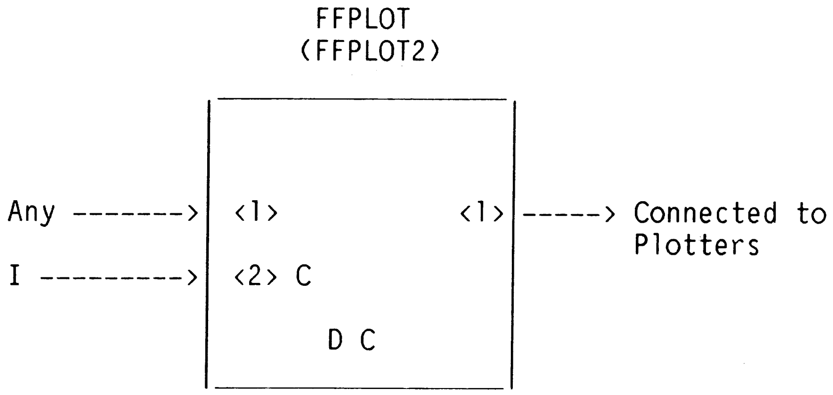
DESCRIPTION

INPUT
<1> – TRUE = enable, FALSE = disable (constant)
<2> – Connected to system

OUTPUT
<1> – connected to Terminal Emulator
<2> – not used

NOTES

The **INITialize** command automatically sends a TRUE to input <1> to enable the display of error messages.



PURPOSE

Causes a form feed at the specified plotter.

DESCRIPTION

INPUT
<1> - trigger
<2> - plotter number (constant)

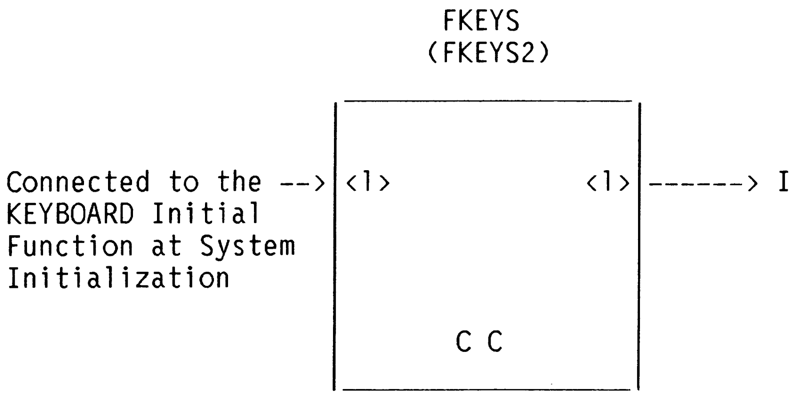
OUTPUT
<1> - connected to plotters

DEFAULT

The default for input <2> is plotter 0.

NOTES

If the form feed attempt fails due to an invalid plotter number or an allocation error, no error message appears. Valid plotter numbers are from 0 to 3.


FKEYS
(FKEYS2)
Connected to the KEYBOARD Initial Function at System Initialization
<1>
<1>
I
C C

## PURPOSE

Converts a character received from a keyboard function key to an integer code.

## DESCRIPTION

INPUT
<1> - connected to KEYBOARD

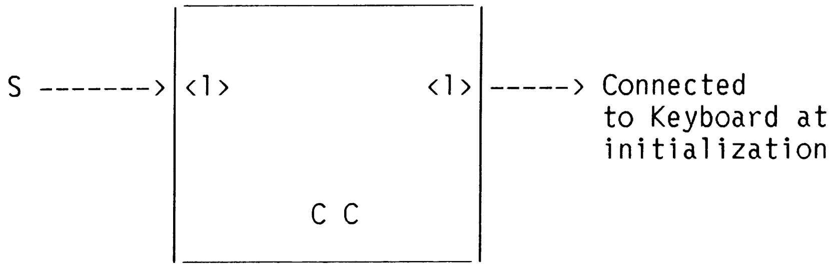
OUTPUT
<1> - integer code (1-36)

## NOTES

Characters are converted as follows:

| Integer Output | Corresponds To |
|---|---|
| 1-12 | Function keys 1-12 |
| 13-24 | Function keys 1-12 with the shift key pressed. |
| 25-36 | Function keys 1-12 with the control key pressed. |

FLABEL0
(FLABEL20)



## PURPOSE

This initial function instance is similar to the **FLABEL1** through **FLABEL12** initial function instances in that it allows the user to specify characters to be displayed in the LED indicators above the function keys. However, unlike **FLABEL1** through **FLABEL12**, which are used to separately specify the 8-character display above each function key, **FLABEL0** allows a single character string (to a maximum of 96 characters) to be specified for display in the twelve 8-character displays. **FLABEL0** treats the 96 LED displays as a single string of characters and spaces.

## DESCRIPTION

INPUT
<1> - string for label

OUTPUT
<1> - connected to keyboard

## NOTES

1. The string of characters on input <1> is displayed starting in the leftmost LED location (the first of the 8 LEDs over the first function key).

2. Allowable characters for the function key LED indicators follow:

   ! " # $ % & ' ( ) * + - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z < [ ] _

NOTES (continued)

3. Lowercase letters are converted to uppercase letters for display. A space may also be specified.

4. Carriage return <CR> and line feed <LF> are not legal characters and cause a message that follows <CR> and/or <LF> to be partially lost or ignored.

## FLABEL1 ... FLABEL12

## Initial Function Instance Output

```
                   FLABEL1 ... FLABEL12
                 (FLABEL21 ... FLABEL212)
                 _____________________
                |                     |
S ---------->   |<1>               <1>|------> S
                |                     |
B ---------->   |<2> C                |
                |                     |
B ---------->   |<3> C                |
                |                     |
                |        D C          |
                |_____________________|
```

PURPOSE

Twelve initial function instances are provided to label the eight LED indicators above each of the twelve function keys. **FLABEL1** is used to label the eight LED indicators associated with the first function key; **FLABEL2** is used to label the eight LED indicators for the second function key; and so on, through **FLABEL12**, which is used to label the eight LED indicators for the twelfth function key.

DESCRIPTION

INPUT
<1> - label message
<2> - blink/no blink (constant)
<3> - center/left justify (constant)

OUTPUT
<1> - string to Function Key LED

NOTES

1. Input <1> accepts a character string (up to eight characters) to be displayed on the corresponding function key LED indicators. The constant Boolean on input <2> selects blink (TRUE) or no blink (FALSE) for the displayed characters. The constant Boolean on input <3> controls whether the displayed message will be centered in the eight available locations (TRUE) or whether the first character will be placed in the leftmost of the eight locations (left justified) (FALSE).

NOTES (continued)

2. If inputs <2> and <3> are not used, the message will not blink and will be centered.

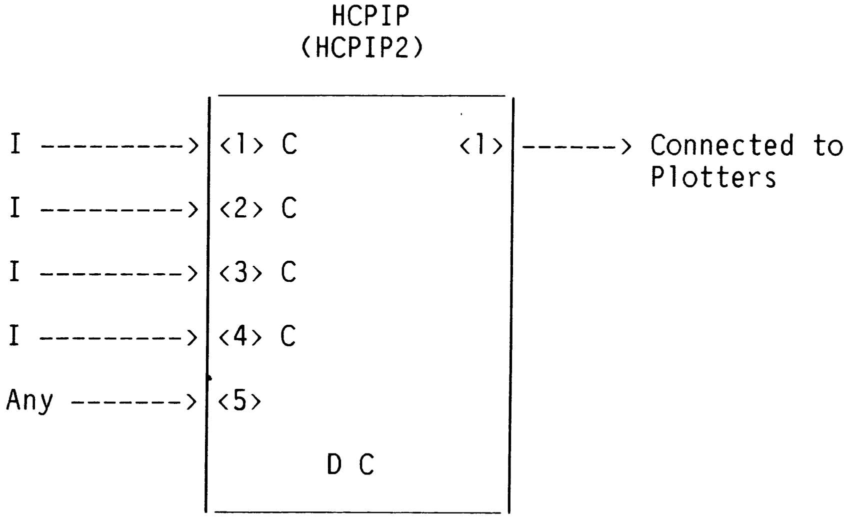
3. Allowable characters for Function Key labels are:

   ! " # $ % & ' ( ) * + - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z < [ ] _

4. Lowercase letters are converted to uppercase letters for display. A space may also be specified.

5. Carriage return <CR> and line feed <LF> are not legal characters and cause a message that follows <CR> and/or <LF> to be partially lost or ignored.

6. The **FLABEL1** through **FLABEL12** function instances are used to separately program the 8-character LED displays associated with a particular function key. If the entire set of 96 LEDs (12 function keys x 8 characters per function key) is to be programmed as a single message, then **FLABEL0** must be used.



## PURPOSE

Initializes plotters so that hardcopies can be obtained of screen displays.

## DESCRIPTION

INPUT
<1> – plotter #0 type (constant)
<2> – plotter #1 type (constant)
<3> – plotter #2 type (constant)
<4> – plotter #3 type (constant)
<5> – trigger

OUTPUT
<1> – connected to plotters

## DEFAULTS

The default for input <1> is 1, the default for inputs <2>, <3>, and <4> is 0.

## NOTES

The initial function instance HCPIP (HardCoPy Initialize Plotters) is automatically triggered at system configuration time during booting. You must also trigger HCPIP after connecting or disconnecting a plotter.

(continued)

## NOTES

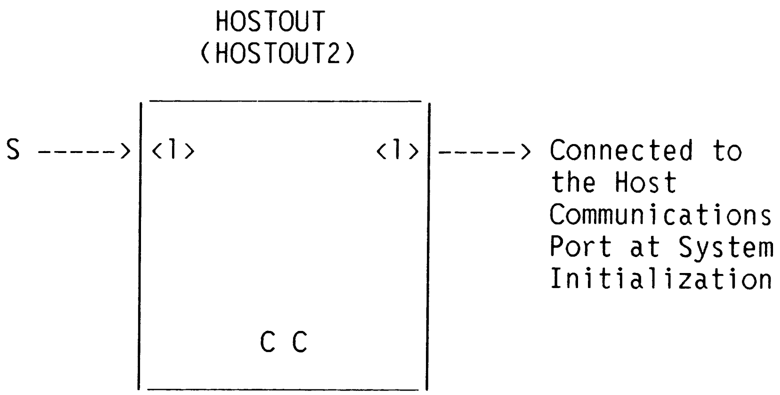
1. Inputs <1>, <2>, <3>, and <4> specify the types of plotters at ports 0, 1, 2, and 3, respectively. There are currently two admissible values for these inputs: 0 (no plotter) and 1 (Versatec V80). The default plotter type at input <1> is 1; the default type for inputs <2>, <3>, and <4> is 0. If invalid values are input, the default value is automatically used without an error message or other notification.

2. Input <5> triggers the function and initializes the plotters.

3. If you trigger HCPIP while a plot is in progress, initialization will not occur.

Initial Function Instance
Output

# HOSTOUT


HOSTOUT
(HOSTOUT2)
S
<1>
<1>
Connected to the Host Communications Port at System Initialization
C C

## PURPOSE

Accepts a string on input <1> and outputs the string for communication to the host. At initialization, the input to HOSTOUT is connected to a network used for the I/O routines.

## DESCRIPTION

INPUT
<1> - string

OUTPUT
<1> - connected to Host Communications Port

```
                         INFORMATION
                        (INFORMATION2)
                     __________________
                    |                  |
                    |    (F:CBROUTE    |
                    |     instance)    |
                    |                  |
        B -------> |<1> C          <1>|----------> Connected to
                    |                  |            Terminal Emulator
                    |                  |            at initialization
  Connected to      |                  |
  system at ------>|<2>            <2>|--------> Not connected
  initialization    |                  |
                    |        D D       |
                    |__________________|
```

PURPOSE

Enables and disables the display of information messages.

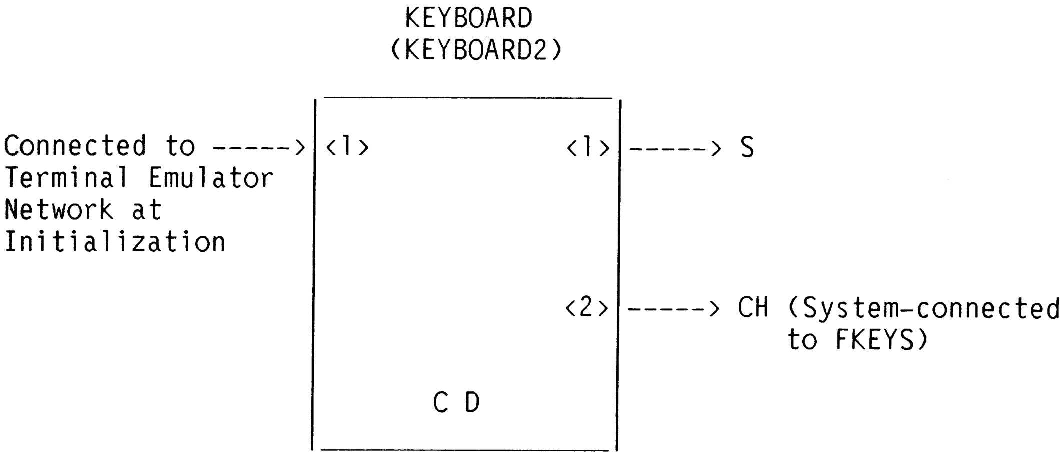
DESCRIPTION

INPUT
<1> - TRUE = enable, FALSE = disable
<2> - connected to system

OUTPUT
<1> - connected to Terminal Emulator
<2> - not used

NOTES

The **INITialize** command automatically sends a TRUE to input <1> to enable the display of information messages.


KEYBOARD
(KEYBOARD2)
Connected to Terminal Emulator Network at Initialization
<1>
<1>
S
<2>
CH (System-connected to FKEYS)
C D

## PURPOSE

Connected at initialization to accept an ASCII character string from the keyboard.

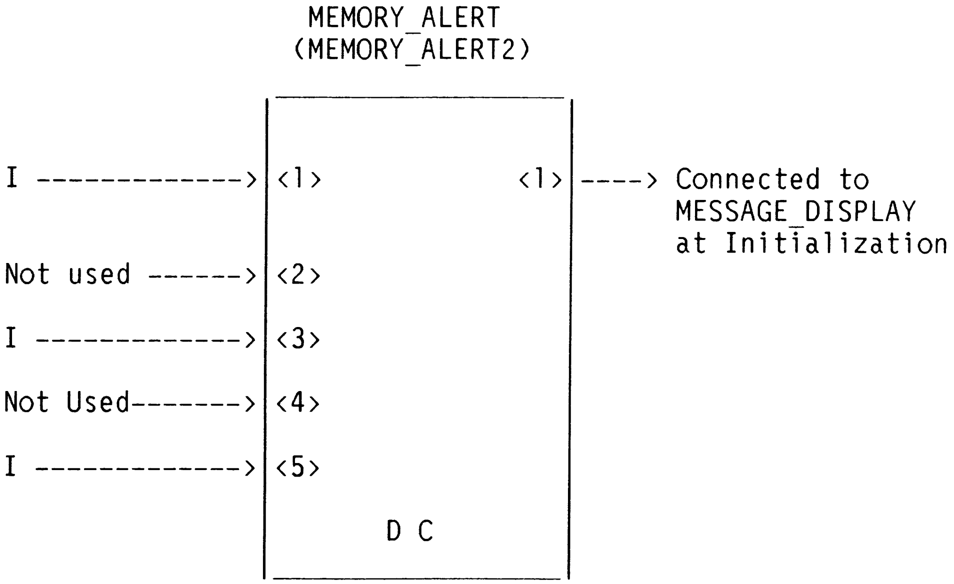
## DESCRIPTION

INPUT
<1> - connected to Terminal Emulator network

OUTPUT
<1> - characters not preceded by CONTROL V
<2> - characters preceding CONTROL V

## NOTES

Input characters are checked for a preceding CONTROL V character. If a character is preceded by a CONTROL V, the CONTROL V is removed by the function and the associated character is output on output <2>, which is system-connected to the input of FKEYS. Characters that are not preceded by a CONTROL V are output on output <1>.



## PURPOSE

Generates a message and a bell alarm when system memory is 75 percent full.

## DESCRIPTION

INPUT
<1> – memory threshold percentage for reporting
<2> – unused
<3> – sampling interval
<4> – unused
<5> – integer specifying memory threshold in bytes for vector list creation

OUTPUT
<1> – connected to MESSAGE_DISPLAY

(continued)

## DEFAULTS

The threshold specified on input <1> is set at 75% unless changed by the user. The sampling interval is set at 10 seconds unless changed. Input <5> defaults to 0.

## NOTES

The number of bytes specified on input <5> is the minimum number of bytes that must be available in memory for the system to create vector list. Once this threshold has been reached, a vector list will be only partially created, or not created at all. When this occurs, the error message "E 105 *** cannot complete operation due to insufficient memory" is issued. This applies to vector lists, characters, labels, polynomials, bsplines, patterned vector lists, and polygons.

1. Memory status is sampled at 10-second intervals. The message displayed is of the form:

   MASS MEMORY nn PERCENT FILLED.

2. If the amount of memory used falls below the threshold, the message is removed.

3. Output <1> is connected to MESSAGE_DISPLAY at initialization.

4. If the user wishes to change the percentage that generates the alarm, another value must be sent to input <1>. If the user wishes to specify a sampling interval other than 10 seconds, another value must be sent to input <3>. The value is an integer that specifies the number of seconds to wait before rechecking memory.

```
                      MEMORY_MONITOR
                     (MEMORY_MONITOR2)
                   ____________________
                  |                    |
I --------------->|<1>              <1>|-----> S and bell
                  |                    |
I --------------->|<2> C            <2>|-----> I
                  |                    |
I --------------->|<3> C            <3>|-----> B
                  |                    |
                  |        D D         |
                  |____________________|
```

## PURPOSE

Notifies the user of the number of bytes that are available for use out of a maximum number of bytes available at system initialization and of the elapsed time since initialization.

## DESCRIPTION

INPUT
<1> – memory threshold percentage
<2> – delta value
<3> – sampling interval

OUTPUT
<1> – message string and bell
<2> – percentage full
<3> – TRUE if threshold is exceeded, not sent if otherwise

## DEFAULTS

The threshold is set at 75%, the delta value is set at 0, and the sampling rate is set at 10 seconds unless changed by the user.

(continued)

## NOTES

1. None of the outputs from this function instance are connected upon system initialization. The user must connect output <1> of MEMORY_MONITOR input <1> of MESSAGE_DISPLAY. This causes the message to be displayed in the message display area of the screen and a bell to be sent to the keyboard. The message displayed is of the form:

   ```
   nnnnnn bytes free out of nnnnnnnn bytes maximum at dd hh:mm:ss
   ```

2. Output <2> is an integer representing the .percentage of memory filled. Unless a change on input <2> (since the last report) is equal to or greater than the previous value on input <2>, no report is given.

3. Output <3> is a Boolean that is output as a TRUE if the threshold indicated on input <1> is crossed.

```
                      MESSAGE_DISPLAY
                     (MESSAGE_DISPLAY2)
                    ____________________
                   |                    |
S ---------------->|<1>              <1>|-----> S and bell
                   |                    |
                   |                    |
                   |        D C         |
                   |____________________|
```

PURPOSE

Displays error messages and informational messages in the MESSAGE_DISPLAY area of the PS 300 display. At initialization, input <1> is connected by the system to output <1> of MEMORY_ALERT and error-detection functions. Output <1> is connected to input <1> of FLABELO so that bell messages can be sent to the keyboard.

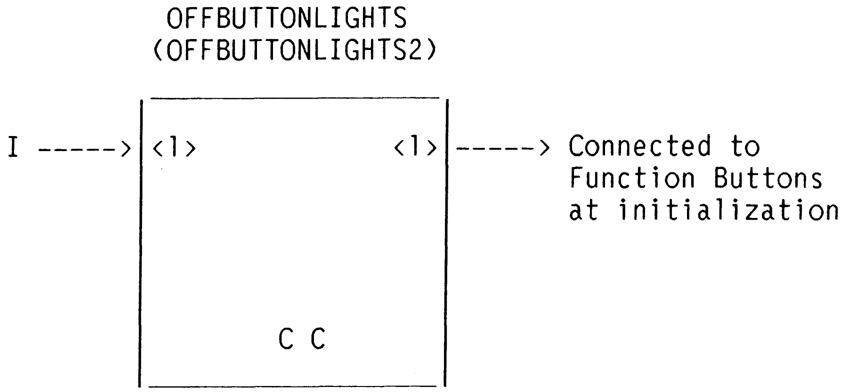
DESCRIPTION

INPUT
<1> - connected to MEMORY_ALERT and error-detection functions

OUTPUT
<1> - string and bell connected to FLABELO

NOTES

1. Each string received is treated as a complete message. Incoming characters are displayed at position 1 and replace the previous message.



## PURPOSE

Turns off lighted buttons on the Function Buttons unit.

## DESCRIPTION

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

OUTPUT
<1> - connected to Function Buttons

## NOTES

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

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

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

## PURPOSE

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

## DESCRIPTION

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

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

(continued)

NOTES

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

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

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

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

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

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

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

NOTES (continued)

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

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

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

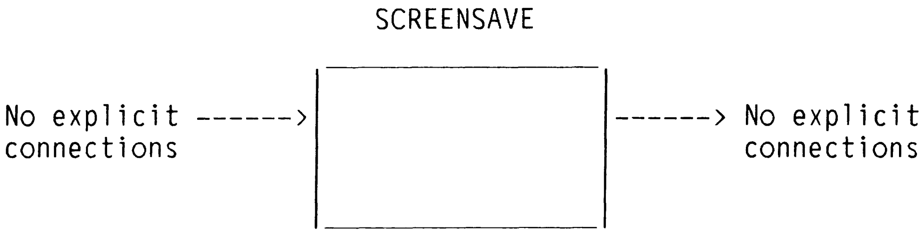
    a. Send an INITialize command. This will remove the PICK function and replace it with a new instance of the PICK function.

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

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

EXAMPLE

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



SCREENSAVE

No explicit connections ------>

------> No explicit connections

PURPOSE

Helps to protect the PS 300 screen from phosphor damage by slowly shifting the viewport in a way that is imperceptible to the user. The viewport moves right two line widths, up two line widths, left two line widths, and down two line widths, and repeats this cycle as long as SCREENSAVE in in effect. SCREENSAVE is on by default.

NOTES

1. Note that SCREENSAVE has no explicit inputs or outputs; the only way to use this function is to instance it when phosphor protection is desired and to delete the instance (using NIL) when it is not desired. To disable screen-saving, enter the command

   ```
   SCREENSAVE := NIL;
   ```

   To enable screen-saving, enter the command

   ```
   SCREENSAVE := F:SCREENSAVE;
   ```

2. Screen-saving should be set to NIL before timed-exposure photographs of the PS 300 screen are taken.

3. Despite SCREENSAVE, users should still exercise other customary precautions against phosphor burn (e.g., avoiding the display of high-intensity images for long periods of time).

```
                          SHADINGENVIRONMENT
                       ______________________
                      |                      |
2D, 3D ------------>  |<1>                <1>|-------> PS 340 Raster Display
                      |                      |
R, 2D, 3D --------->  |<2>                   |
                      |                      |
3D ---------------->  |<3>                   |
                      |                      |
R ----------------->  |<4>                   |
                      |                      |
I ----------------->  |<5>                   |
                      |                      |
R ----------------->  |<6>                   |
                      |                      |
B ----------------->  |<7>                   |
                      |        D C           |
                      |______________________|
```

PURPOSE

For use with the PS 340 system, this function allows you to control various non-dynamic factors of shaded renderings displayed on the raster screen.

DESCRIPTION

INPUT
<1> - ambient color
<2> - background color
<3> - raster viewport
<4> - exposure
<5> - quality level
<6> - depth cueing
<7> - screen wash

OUTPUT
<1> - connected to the PS 340 Raster Display

(continued)

NOTES

1. Ambient color: input <1> accepts a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation, and intensity, to specify the ambient color. The ambient color is combined with the result obtained from the light sources to determine the color of ambient light. The default ambient color is white, with a default intensity of .25. The ambient color is analagous to the color reflected off a wall.

2. Background color: input <2> accepts a real number as hue, a 2D vector as hue and saturation, and or a 3D vector as hue, saturation, and intensity to specify the background color. The raster screen will be colored with the background color prior to any shaded rendering. The default background color is black (0,0,0).

3. Raster viewport: input <3> accepts a 3D vector as the viewport on the raster image buffer where shaded renderings will be displayed. Raster viewports are always square, the lower left corner being given by the X and Y coordinates of the vector, and its size given by the Z coordinate, such that the upper right corner is at (x+z,y+z). Values are rounded to the nearest pixel. The default viewport is (80,0,480). The viewport is not intended for magnification of small parts of the calligraphic image, but for mapping the square vector display onto the rectangular raster display.

   The viewport is also intended to allow multiple images to be generated side by side on the raster display. Thus, the largest recommended value for the viewport is (0,-80,640). The actual largest viewport is somewhat larger and depends on combinations of the three values. The image is clipped to the physical raster for which $0 \le X < 640$ and $0 \le Y < 480$.

4. Exposure: input <4> accepts a real number as the exposure, controlling the overall brightness of the picture. The exposure is like that on a camera. If a picture is taken of an object with a very bright specular highlight, it may be so bright that the rest of the object is darkened. If three light sources exist, the object would be about three times brighter, making the object too bright. The exposure should be brought down to control this.

NOTES (continued)

The exposure is multiplied by the intensity at each pixel and the result clipped to the maximum intensity. This enables the overall brightness of a rendering to be increased without causing bright spots to exceed maximum intensity (instead forming "plateaus" of maximum intensity). Note that this may cause changes in color on a plateau, where color has reached its maximum, but the others have not. Exposure values may vary between .3 and 3, values outside that range being changed to .3 or 3. The default exposure is 1.

5. Quality level: input <5> accepts an integer as quality level. The quality controls the number of pixels over which filtering applied. Jagged edges are characteristic of a raster display, so the fuzzier the edges, the better quality the picture. Values of 1, 3, 5, and 7 are allowed, meaning that the effect of coloring a pixel will be spread over a square of pixels with that number on a side, centered on the colored pixel. Because of anti-aliasing, pictures are good at quality 1. (The default value 1 is the typical choice.) Values of 3, 5, and 7 produce better quality renderings in terms of anti-aliasing but are time-consuming to process.

6. Depth cueing: input <6> accepts a real number in the range of 0 to 1 to control depth cueing in the shaded image (0 specifying no depth cueing and 1 specifying maximum depth cueing). As perceived depth from the viewer increases, the intensity of the colors decreases from maximum (1) at the nearest point to the given proportion of maximum at the farthest. Thus 0 gives a ramp ending in black at the back clipping plane, while 1 turns off the effect of depth cueing. The default is 0.2 giving a fairly large depth cueing effect.

7. Screen wash: input <7> accepts a Boolean, and is the only input to cause a visual effect immediately. True causes the whole physical raster screen to be filled with the current background color, while false just fills the currently defined viewport (clipped to the screen).

```
                            SPECKEYS
                           (SPECKEYS2)
                     |-----------------|
Connected to ------->|<1>           <1>|-------> S
Terminal Emulator    |                 |
Network at           |                 |
Initialization       |       C C       |
                     |-----------------|
```

PURPOSE

Connected at initialization to accept an ASCII character string from the keyboard. Input characters are checked for a preceding "control V" character. If a character is preceded by a "control V", SPECKEYS removes the "control V" and outputs the associated character on output <1>. (Characters not preceded by a "control V" appear at the output of KEYBOARD instead.)

DESCRIPTION

INPUT
<1> - connected to Terminal Emulator

OUTPUT
<1> - string

NOTES

Note that neither SPECKEYS nor KEYBOARD outputs function key values. The initial function FKEYS supplies these values.

```
                      TABLETIN
                     (TABLETIN2)
                  _______________
R -------------> |<1>C        <1>| ------> 2D
                 |               |
S Connected ---> |<2>         <2>| ------> B
  to data        |               |
  tablet at      |            <3>| ------> I
  initialization |               |
  (string)       |            <4>| ------> B
                 |               |
R -------------> |<3> C       <5>| ------> B
                 |               |
I -------------> |<4>         <6>| ------> 2D
                 |      D D      |
                 |_______________|
```

## PURPOSE

Connected at system initialization to accept data from the data tablet on input <2>. This data includes 2D vectors, an indication of the open/closed condition of the stylus tipswitch (or 4-button cursor), and an indication of the switch number for systems using a 4-button cursor instead of a stylus.

## DESCRIPTION

INPUT

<1> - delta x, y (constant)
<2> - string (system connected to Tablet)
<3> - tablet size (constant)
<4> - wait time

OUTPUT

<1> - x,y coordinate (position/line)
<2> - TRUE = switch closed, FALSE = switch open
<3> - switch number
<4> - tipswitch transition
<5> - range transition
<6> - x,y when switch closed

(continued)

## DEFAULT

The default delta x,y on input <1> is .002. The default tablet size on input <3> is 2200. The default wait time on input <4> is 8 centiseconds.

## NOTES

1. Input <1> accepts a real number that specifies the minimum change in X or Y required on input <2> before output <1> is sent. The default value is .002.

2. Input <3> accepts an integer that specifies the number of points full-scale for the data tablet being used. The default value is 2200, corresponding to the standard 11-inch x 11-inch data tablet.

3. Input <4> is a wait time for the data tablet in centiseconds; a FALSE is sent on output <5> if the tablet stops sending data for longer than this duration. The default value is 8. It should never be necessary to **SEND** to this input, since **TABLETOUT** sends an appropriate value here automatically (see **TABLETOUT**).

4. The Boolean on output <2> indicates the condition of the stylus tipswitch (or cursor button) as follows:

   TRUE = Stylus tipswitch closed or cursor button pressed.
   FALSE = Stylus tipswitch open or cursor button not pressed.

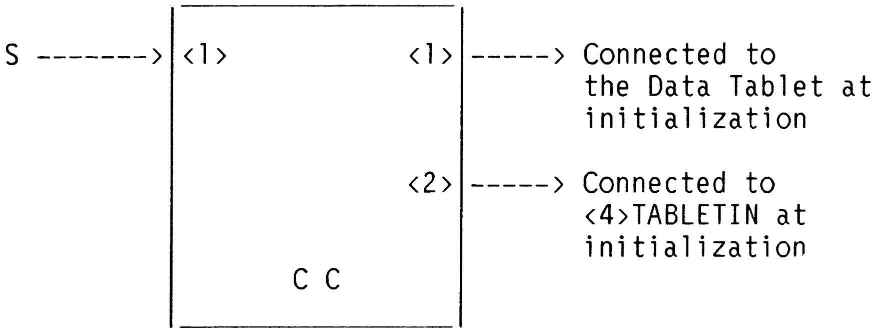
5. The integer on output <3> is the sum of the numbers of the pressed buttons on the 4-button cursor. The buttons are numbered 1, 2, 4, and 8. If button 1 and button 4 are pressed simultaneously, 5 is output.

6. A TRUE appears at output <4> whenever the tipswitch goes from open to closed, and a FALSE whenever the tipswitch goes from closed to open. For button-type cursors, output <4> is TRUE when a button is pushed and FALSE when the button is released.

7. Output <5> indicates transitions in stylus proximity (i.e., from "receiving data" to "not receiving data" and vice versa). A TRUE appears here when data is received from the tablet after a period of no data. A FALSE is sent when data does not arrive from the tablet in time. The time is the number of hundredths of a second specified at input <4>.

8. Output <6> is the (x,y)-position of the stylus when the tipswitch goes from open to closed.

TABLETOUT
(TABLETOUT2)


S
<1>
<1>
Connected to the Data Tablet at initialization
<2>
Connected to <4>TABLETIN at initialization
C C

PURPOSE

Provides the means to set operating parameters in the Data Tablet by sending a character to input <1>. The character also determines a value to be sent to <4>**TABLETIN**, setting the tablet's timeout interval. If a multi-character string is sent to input <1>, only the final character of the string is used.

DESCRIPTION

INPUT
<1> – character or string

OUTPUT
<1> – connected to Data Tablet
<2> – connected to <4>**TABLETIN**

(continued)

NOTES

1. Characters for mode settings are as follows:

| CHARACTER | MODE | SAMPLING RATE | TIMEOUT INTERVAL (IN CENTISECONDS) |
|---|---|---|---|
| S | Stop | Idle | |
| P | Point* | Manual control | |
| @ | Switched Stream** | 2 | 52 |
| A | . | 4 | 27 |
| B | . | 10 | 12 |
| C | . | 20 | 8 |
| D | . | 35 | 5 |
| E | Switched Stream | 70 | 3 |
| H | Stream*** | 2 | 52 |
| I | . | 4 | 27 |
| J | . | 10 | 12 |
| K | . | 20 | 8 (default) |
| L | . | 35 | 5 |
| M | Stream | 70 | 3 |

* Pressing the stylus on the tablet or the button on the cursor sends out the single x,y coordinate pair.

** Pressing the stylus on the tablet surface or the button on the cursor causes x,y coordinate pairs to be output continuously at the selected sampling rate until the stylus is lifted or the cursor button is released.

*** x,y coordinate pairs are generated continuously at the selected sampling rate when the stylus or cursor is in the proximity of the tablet surface. Pressing the stylus on the tablet surface or pressing the cursor button sets the flag character (F) in the output stream.

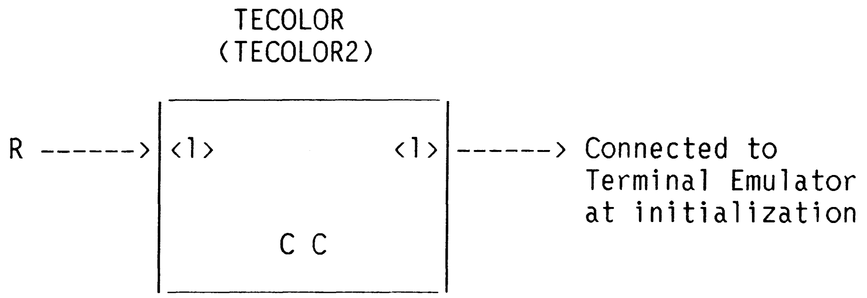
NOTES (continued)

2. Users who have early versions of the PS 300 hardware may not see a cursor from the data tablet after booting. If this is the case, press the RESET button on the back of the tablet immediately following power-up. If the RESET button is not pressed immediately following power-up, the user can later press the RESET button and then enter the command:

   ```
   SEND 'K' TO <1> TABLETOUT;
   ```

   Optionally, a 'J' may be used. Anything greater than 'K' is not recommended.

3. For additional information on the data tablet, refer to the Bit Pad One User's Manual by Summagraphics Corporation, which is included in the customer installation package.



## PURPOSE

Specifies the hue of Terminal Emulator and Setup output on systems with the CSM Calligraphic Display option.

## DESCRIPTION

INPUT
<1> - hue

OUTPUT
<1> - connected to Terminal Emulator

## DEFAULT

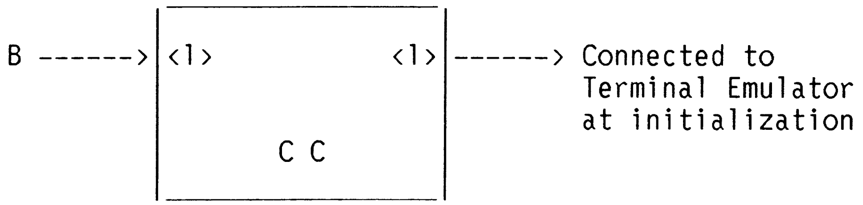
The default hue is 240, pure green.

## NOTES

1. The range of acceptable values is the 0-360 "color wheel" used by the **SET COLOR** command, in which 0 represents pure blue, 120 pure red, and 240 pure green. The default is 240. Out-of-range values are clamped to the nearest in-range value (0 or 360 -- hence always blue).

2. On systems without a CSM Calligraphic Display, **TECOLOR** accepts real values but has no effect.

TSCSM
(TSCSM2)



## PURPOSE

Sets the CSM on or off for the Terminal Emulator

## DESCRIPTION

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

OUTPUT
<1> - connected to Terminal Emulator

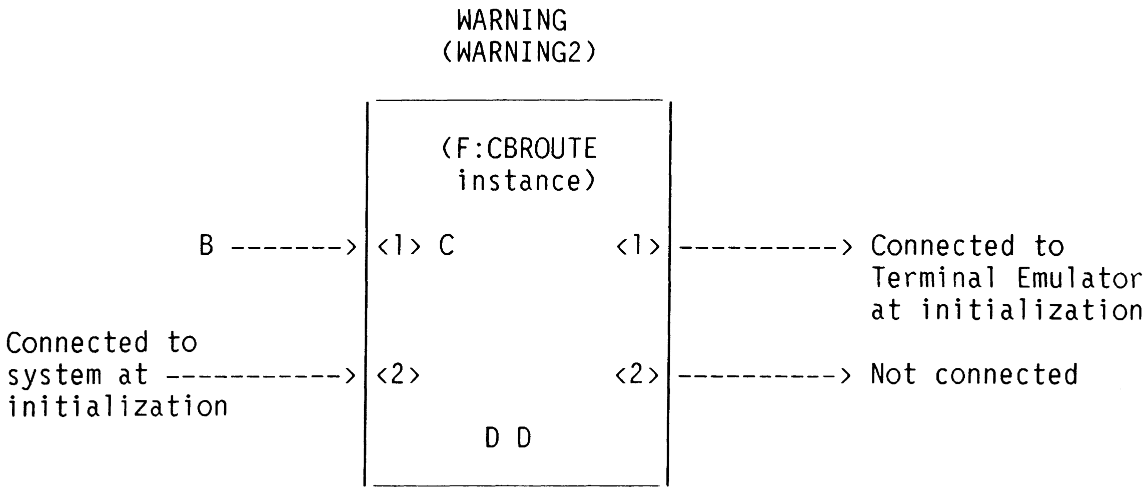
## DEFAULT

The default is FALSE, setting the CSM off.

## NOTES

1. This setting has important consequences for both CSM Calligraphic and monochrome displays! Refer to Section 5.2.5 of the PS 300 User's Manual for guidelines on setting CSM mode ON and OFF.

# WARNING



## PURPOSE

Enables and disables the display of warning messages.

## DESCRIPTION

INPUT
<1> – enable/disable warning messages (constant)
<2> – connected to system

OUTPUT
<1> – connected to Terminal Emulator
<2> – not used

## NOTES

1. A TRUE at input <1> enables warning messages. A FALSE at input <1> disables them. The **INITialize** command automatically sends a TRUE to input <1> enabling the display of warning messages.

## CURSOR
## (CURSOR2)

```
Cursor := VECTOR_LIST ITEMIZED N = 4
          P .035,.035 L -.035,-.035
          P -.035,.035 L .035,-.035;
```

### PURPOSE

This initial structure is a vector list as shown above, which creates a displayable cursor in the form of a cross when the system is initialized.

### DESCRIPTION

INPUT
Vector list

OUTPUT
Displayable "X"-shaped cursor

### NOTES

1. The cursor is controlled by a function network which positions it on the PS 300 screen in response to stylus movement over the data tablet surface. The intensity of the cursor increases when the stylus tip switch is pressed down.

2. The user is free to redefine CURSOR using any other vector list.

# PICK_LOCATION
# (PICK_LOCATION2)

## PURPOSE

PICK_LOCATION is the name assigned at initialization to the system-created picking location.

## DEFAULT

At system initialization, the pick location is defined as the center of the cursor.

## NOTES

The initial **TABLETIN** function instance is connected to PICK_LOCATION and the system-initialized **CURSOR** points to its center.

# APPENDIX A. FUNCTIONS BY CATEGORY

## INTRINSIC FUNCTIONS

### Arithmetic and Logical

F:ACCUMULATE
F:ADD
F:ADDC
F:AND
F:ANDC
F:AVERAGE
F:CDIV
F:CMUL
F:CSUB
F:DIV
F:DIVC
F:MOD
F:MODC
F:MUL
F:MULC
F:NOT
F:OR
F:ORC
F:ROUND
F:SINCOS
F:SQROOT
F:SUB
F:SUBC
F:XOR
F:XORC

### Character Transformation

F:CROTATE
F:CSCALE

## INTRINSIC FUNCTIONS (continued)

### Comparison

F:CGE
F:CGT
F:CLE
F:CLT
F:COMP_STRING
F:EQ
F:EQC
F:GE
F:GEC
F:GT
F:GTC
F:LE
F:LEC
F:LT
F:LTC
F:NE
F:NEC

### Data Conversion

F:CEILING
F:CHARCONVERT
F:CVEC
F:FIX
F:FLOAT
F:MATRIX2
F:MATRIX3
F:MATRIX4
F:PARTS
F:PRINT
F:STRING_TO_NUM
F:TRANS_STRING
F:VEC
F:VECC
F:XFORMDATA
F:XVECTOR
F:YVECTOR
F:ZVECTOR

### Data Selection

F:INPUTS_CHOOSE(n)

## INTRINSIC FUNCTIONS (continued)

### Data Selection and Manipulation

F:ATSCALE
F:BOOLEAN_CHOOSE
F:BROUTE
F:BROUTEC
F:CBROUTE
F:CCONCATENATE
F:CHARMASK
F:CONCATENATE
F:CONCATENATEC
F:CONSTANT
F:CROUTE(n)
F:DELTA
F:FIND_STRING
F:GATHER_STRING
F:LABEL
F:LBL_EXTRACT
F:LENGTH_STRING
F:LIMIT
F:LINEEDITOR
F:MCONCATENATE(n)
F:PASSTHRU(n)
F:PUT STRING
F:RANGE_SELECT
F:ROUTE(n)
F:ROUTEC(n)
F:SEND
F:SPLIT
F:TAKE_STRING
F:VEC_EXTRACT

### Miscellaneous

F:COLOR
F:EDGE_DETECT
F:FETCH
F:NOP
F:PICKINFO
F:POSITION_LINE
F:SYNC(n)

## INTRINSIC FUNCTIONS (continued)

### Object Transformation
F:DSCALE
F:DXROTATE
F:DYROTATE
F:DZROTATE
F:XROTATE
F:YROTATE
F:ZROTATE
F:SCALE

### Timing
F:CLCSECONDS
F:CLFRAMES
F:CLTICKS
F:TIMEOUT

### Viewing Transformation
F:FOV
F:LOOKAT
F:LOOKFROM
F:WINDOW

## INITIAL FUNCTION INSTANCES

### Input
BUTTONSIN
DIALS
KEYBOARD
PICK
SPECKEYS
TABLETIN
TABLETOUT

### Miscellaneous
ERROR
INFORMATION
MEMORY_ALERT
MEMORY_MONITOR
MESSAGE_DISPLAY

## INITIAL FUNCTION INSTANCES (continued)

### Miscellaneous (continued)

SCREENSAVE
SHADINGENVIRONMENT
TECOLOR
TSCSM
WARNING

### Output

CLEAR_LABELS
DLABEL1 ... DLABEL8
DSET1 ... DSET8
FFPLOT
FKEYS
FLABEL0
FLABEL1 ... FLABEL12
HCPIP
HOSTOUT
OFFBUTTONLIGHTS

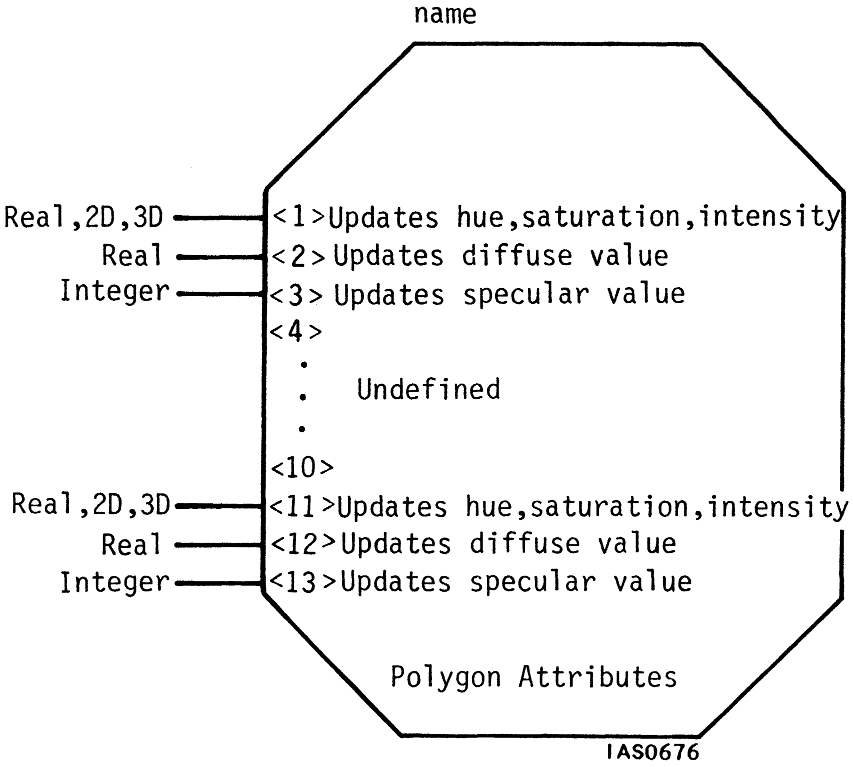
## INITIAL STRUCTURES

CURSOR
PICK_LOCATION

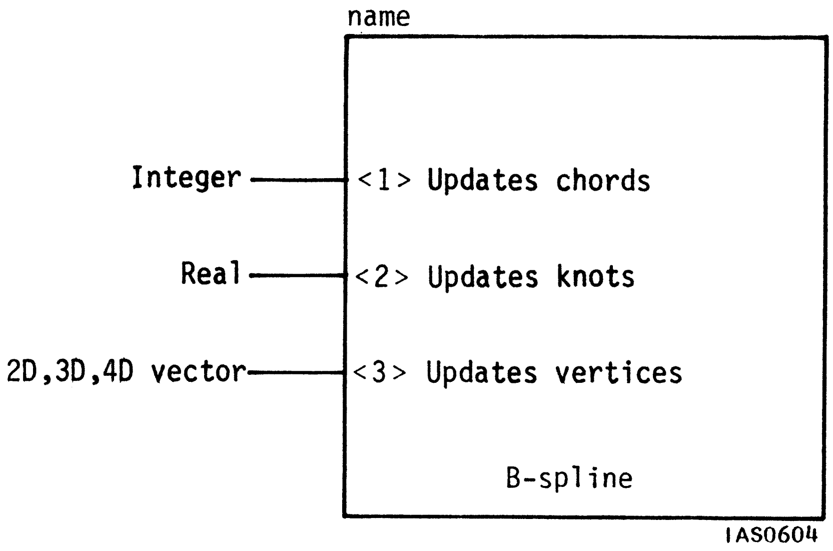
# APPENDIX B. INPUTS TO NODES

This appendix lists nodes in a display which contain data that can be updated using function networks. The name of the PS 300 command which creates the node is given. A diagram shows the number of inputs to the node and the types of data those inputs accept.

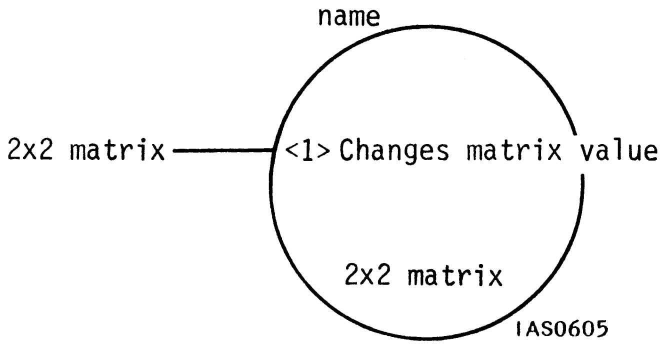
## ATTRIBUTES


name
Real,2D,3D
Real
Integer
<1>Updates hue,saturation,intensity
<2>Updates diffuse value
<3> Updates specular value
<4>
Undefined
<10>
Real,2D,3D
Real
Integer
<11>Updates hue,saturation,intensity
<12>Updates diffuse value
<13>Updates specular value
Polygon Attributes
IAS0676

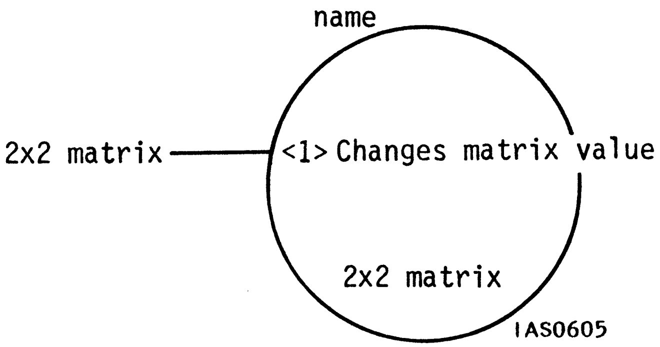
## BSPLINE



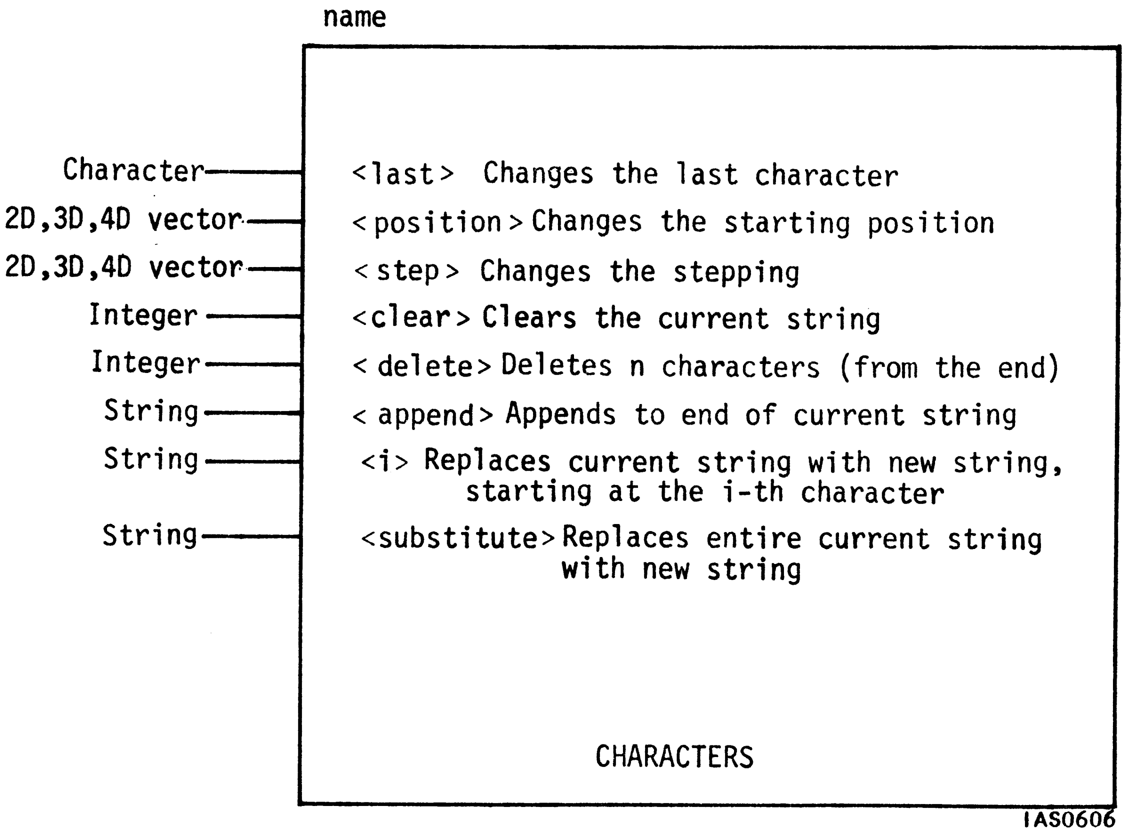
## CHARACTER ROTATE



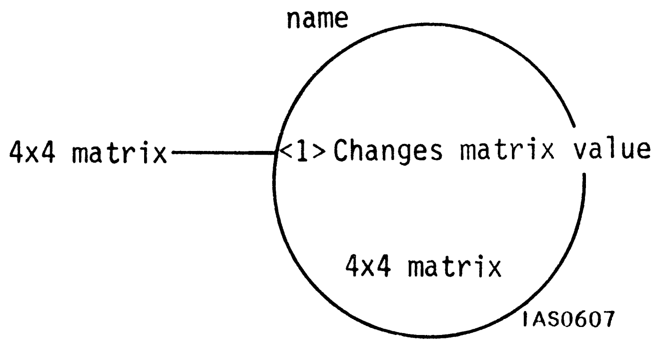
## CHARACTER SCALE



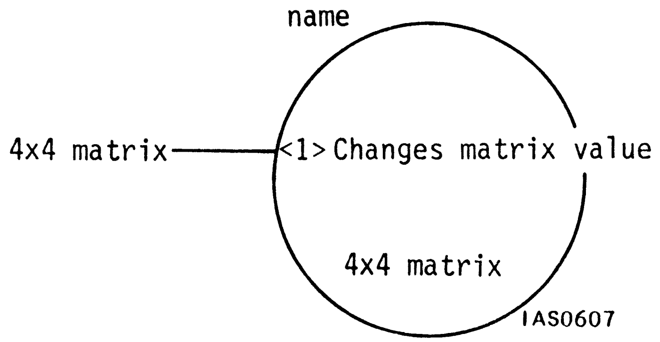
## CHARACTERS


name
Character
2D,3D,4D vector
2D,3D,4D vector
Integer
Integer
String
String
String
<last> Changes the last character
<position> Changes the starting position
<step> Changes the stepping
<clear> Clears the current string
< delete> Deletes n characters (from the end)
< append> Appends to end of current string
<i> Replaces current string with new string, starting at the i-th character
<substitute> Replaces entire current string with new string
CHARACTERS
IAS0606

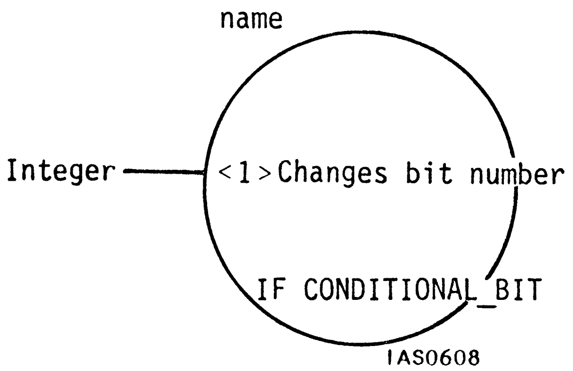
## EYE



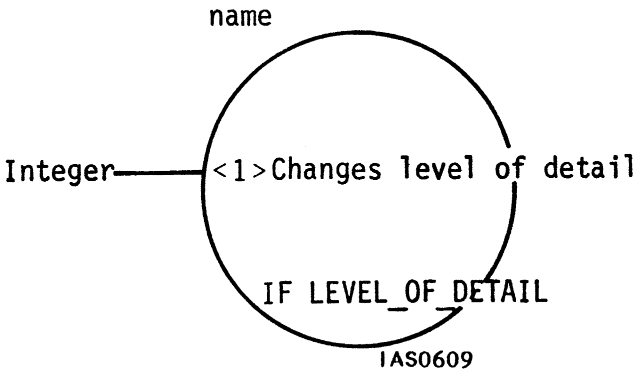
## FIELD_OF_VIEW



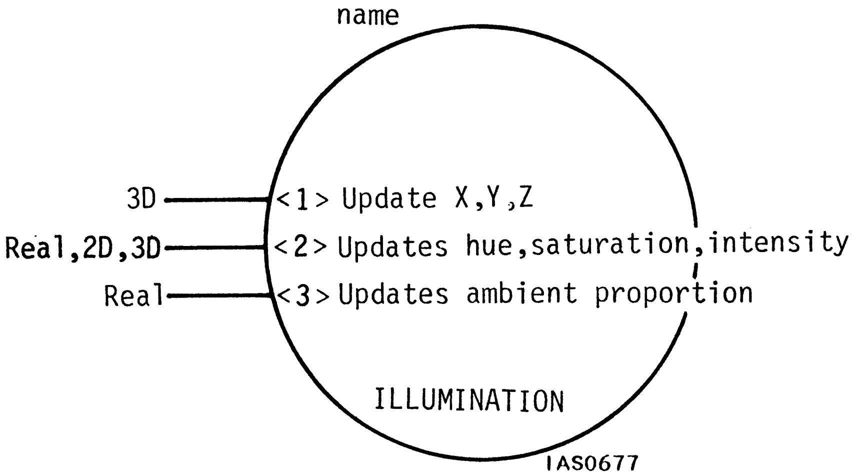
## IF CONDITIONAL_BIT



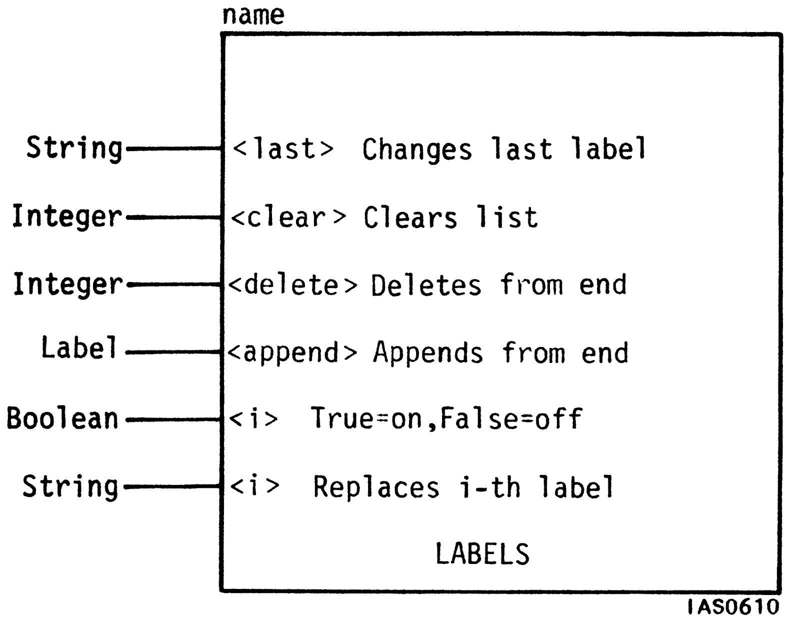
IF LEVEL_OF_DETAIL



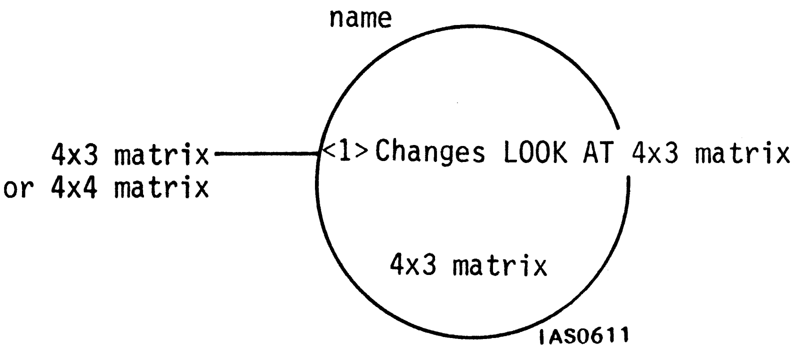
ILLUMINATION


name
3D
Real,2D,3D
Real
<1> Update X,Y,Z
<2> Updates hue,saturation,intensity
<3> Updates ambient proportion
ILLUMINATION
IAS0677

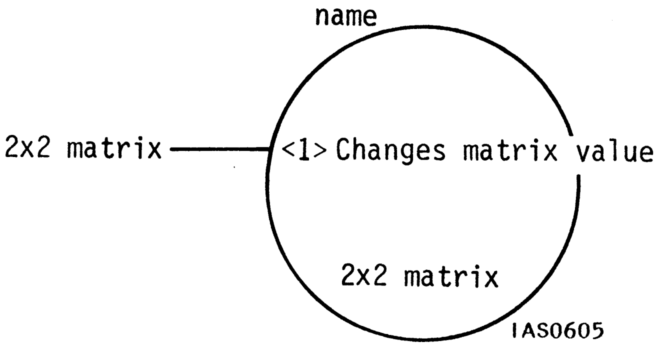
## LABELS


name
String
Integer
Integer
Label
Boolean
String
<last> Changes last label
<clear> Clears list
<delete> Deletes from end
<append> Appends from end
<i> True=on,False=off
<i> Replaces i-th label
LABELS
IAS0610

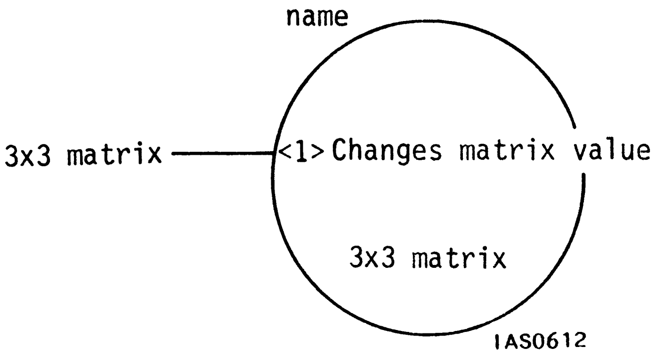
## LOOK


name
4x3 matrix
or 4x4 matrix
<1> Changes LOOK AT
4x3 matrix
4x3 matrix
IAS0611

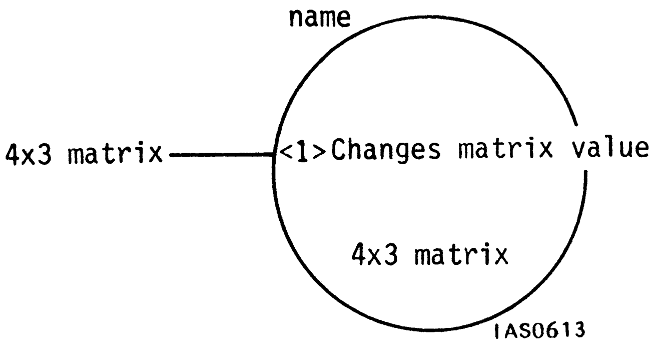
MATRIX_2X2



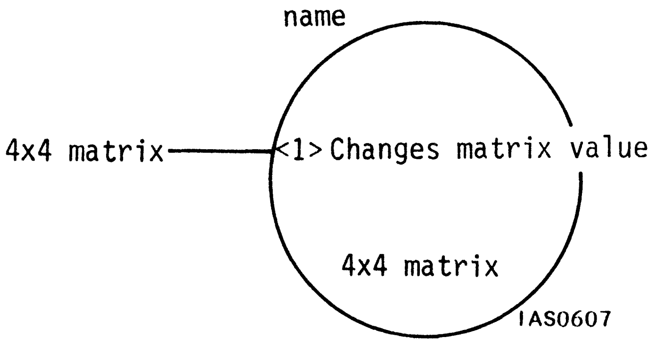
MATRIX_3X3



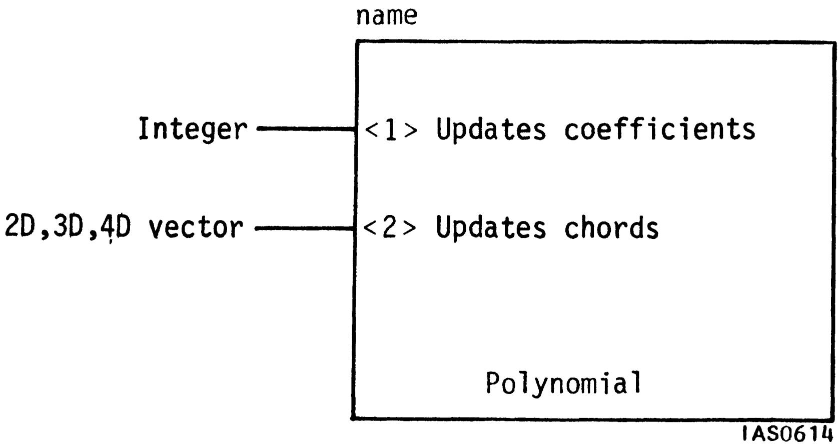
MATRIX_4X3



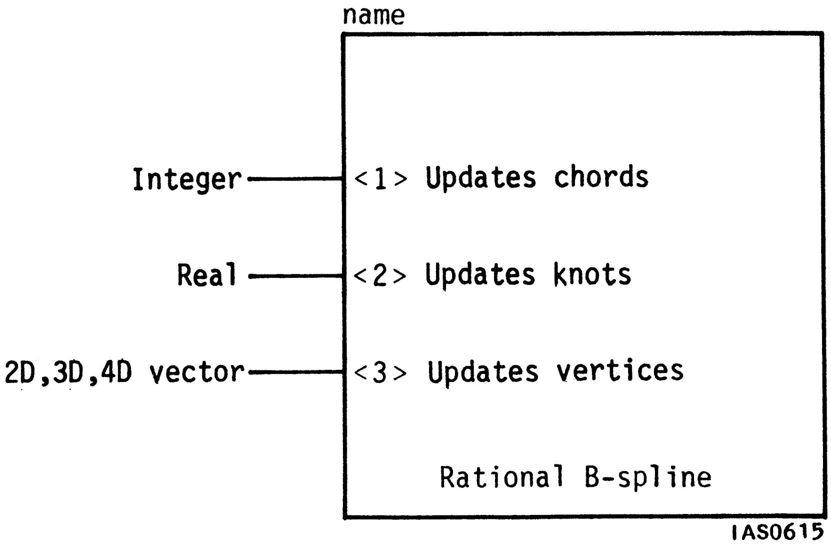
## MATRIX_4X4



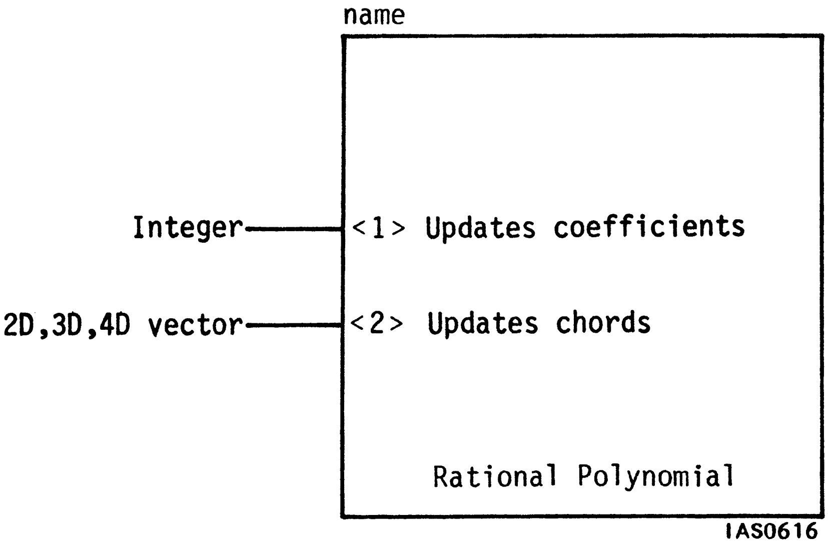
## POLYNOMIAL



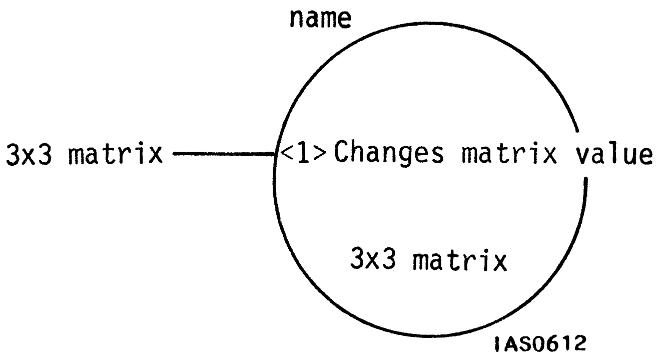
RATIONAL BSPLINE



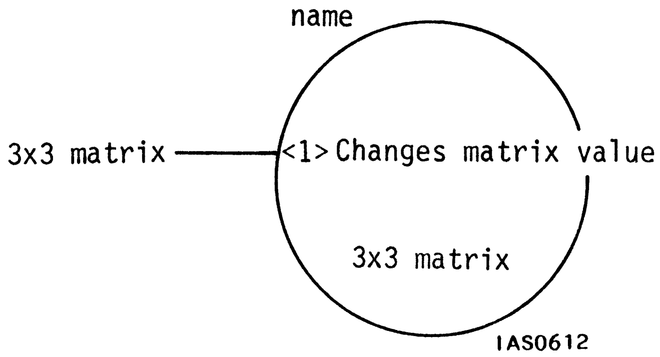
RATIONAL POLYNOMIAL



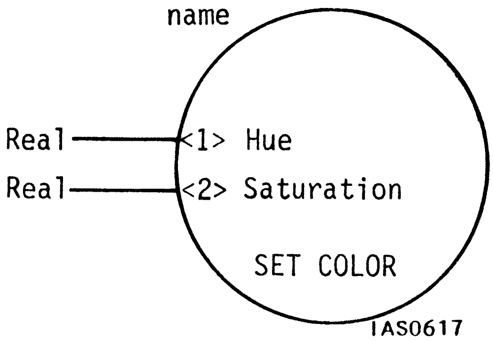
ROTATE



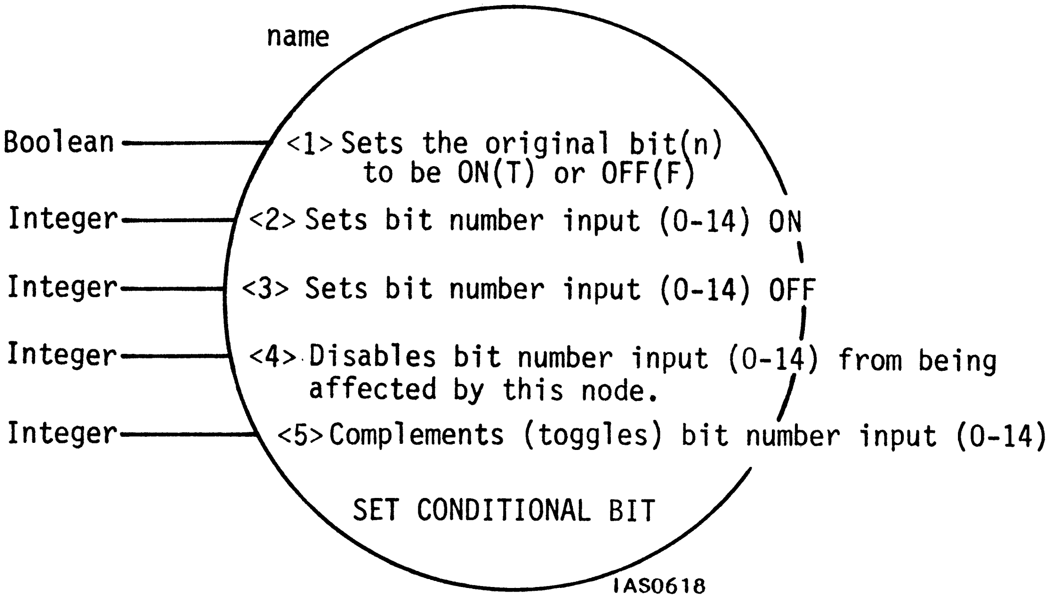
SCALE



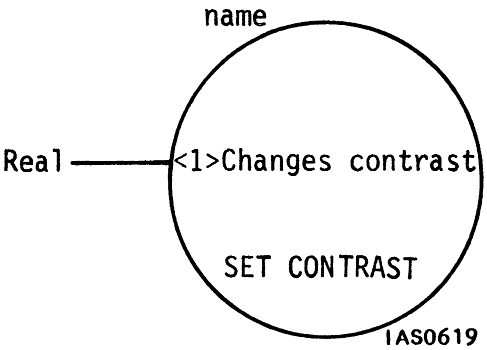
SET COLOR



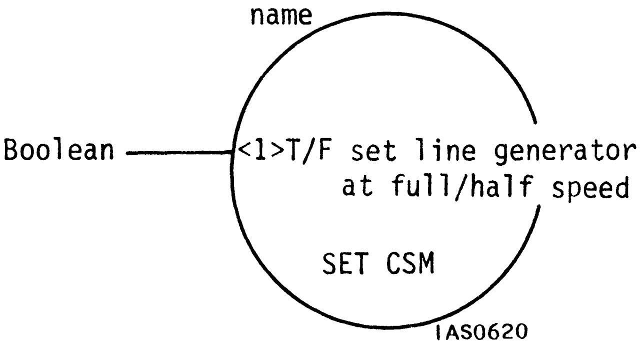
SET CONDITIONAL_BIT


name
Boolean
<1> Sets the original bit(n) to be ON(T) or OFF(F)
Integer
<2> Sets bit number input (0-14) ON
Integer
<3> Sets bit number input (0-14) OFF
Integer
<4> Disables bit number input (0-14) from being affected by this node.
Integer
<5> Complements (toggles) bit number input (0-14)
SET CONDITIONAL BIT
IAS0618

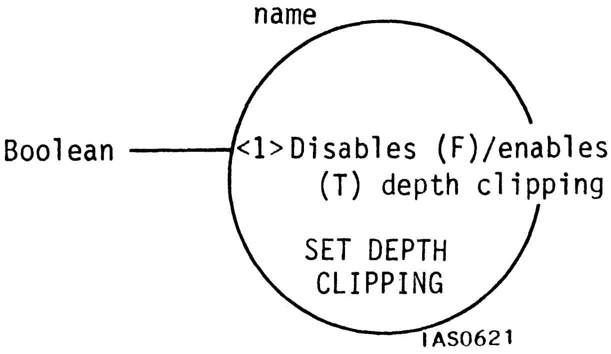
SET CONTRAST



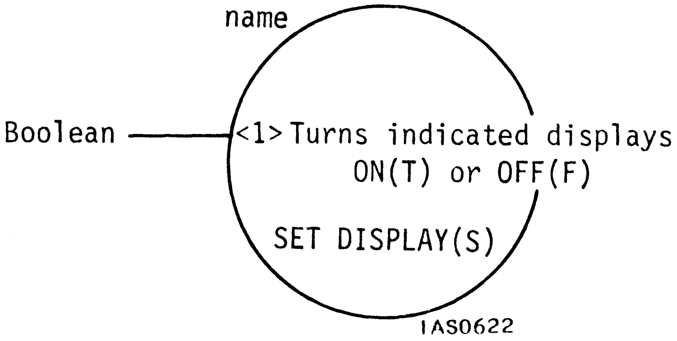
SET CSM



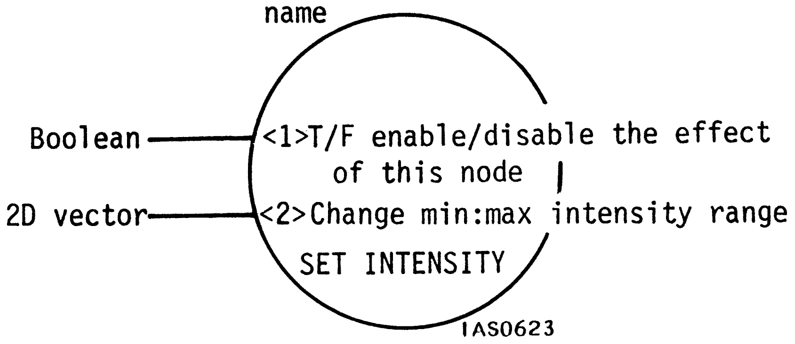
SET DEPTH_CLIPPING



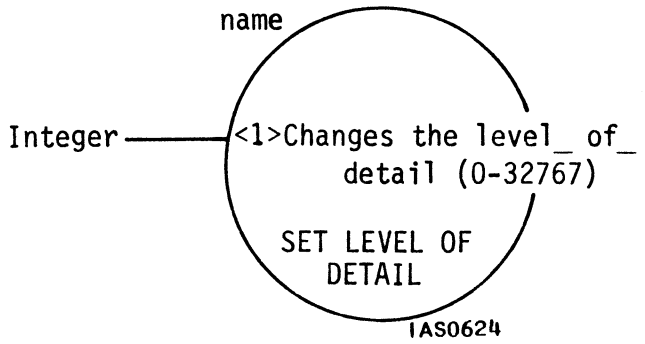
SET DISPLAYS



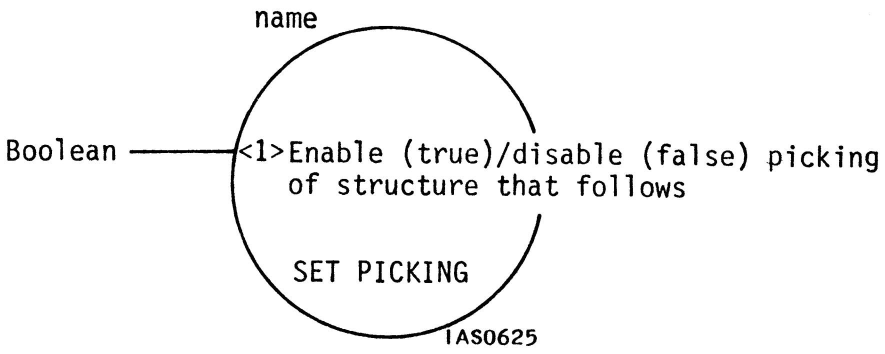
SET INTENSITY


name
Boolean
2D vector
<1>T/F enable/disable the effect of this node
<2>Change min:max intensity range
SET INTENSITY
IAS0623

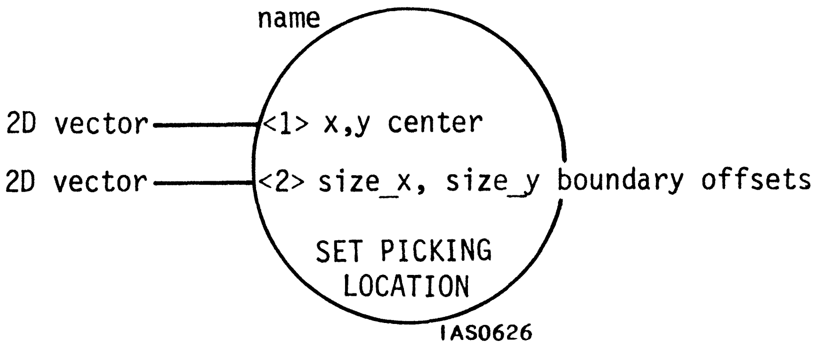
SET LEVEL_OF_DETAIL



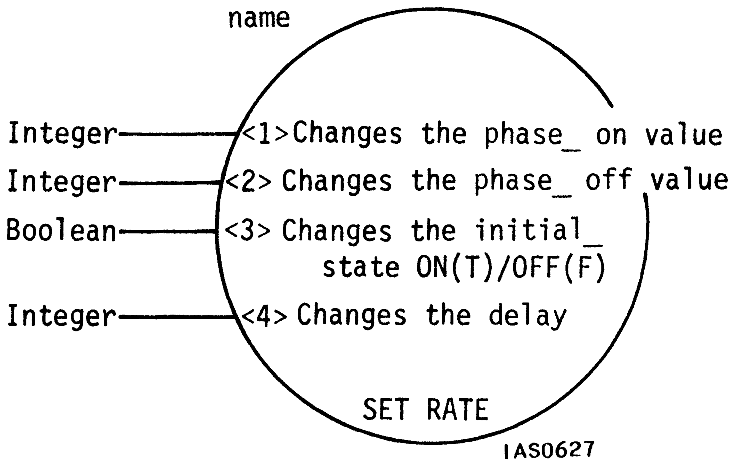
SET PICKING


name
Boolean
<1>Enable (true)/disable (false) picking of structure that follows
SET PICKING
IAS0625

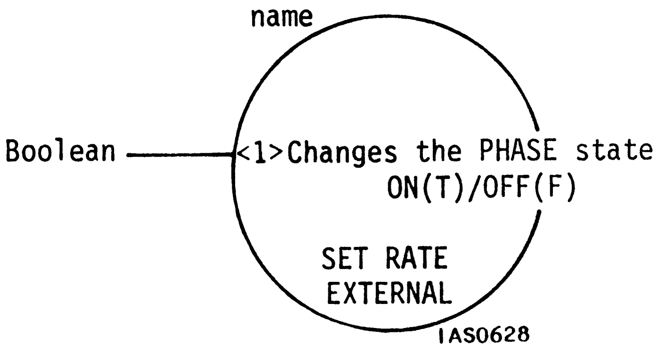
## SET PICKING LOCATION



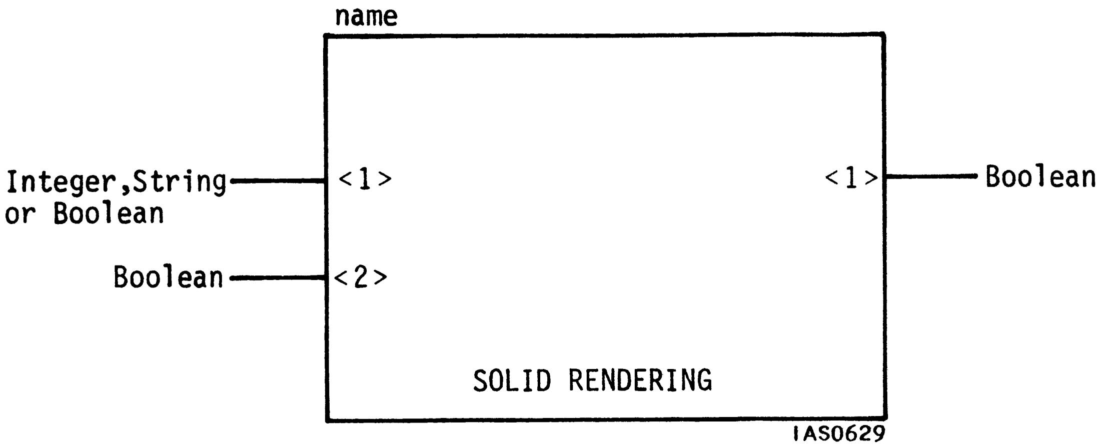
## SET RATE



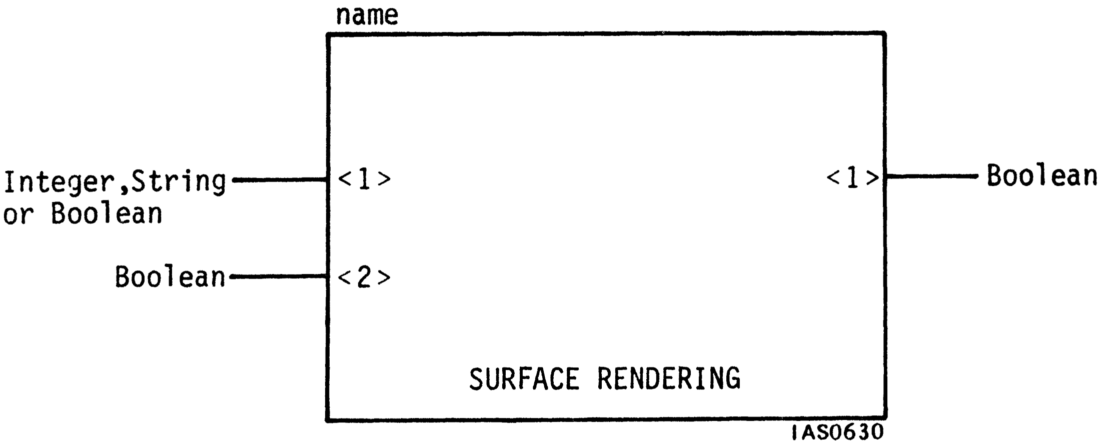
SET RATE EXTERNAL



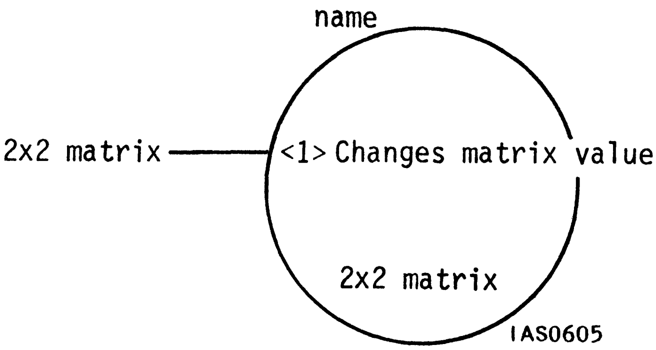
SOLID_RENDERING



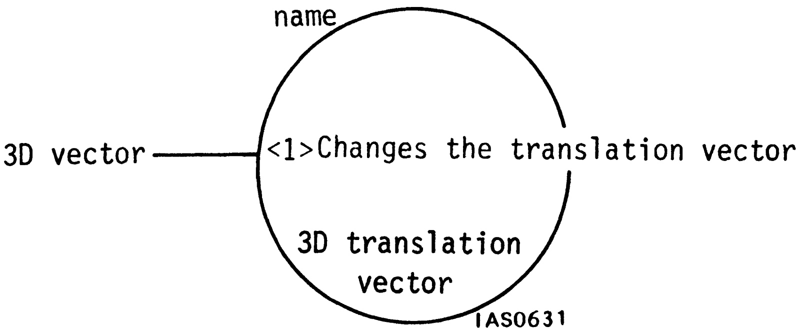
SURFACE_RENDERING



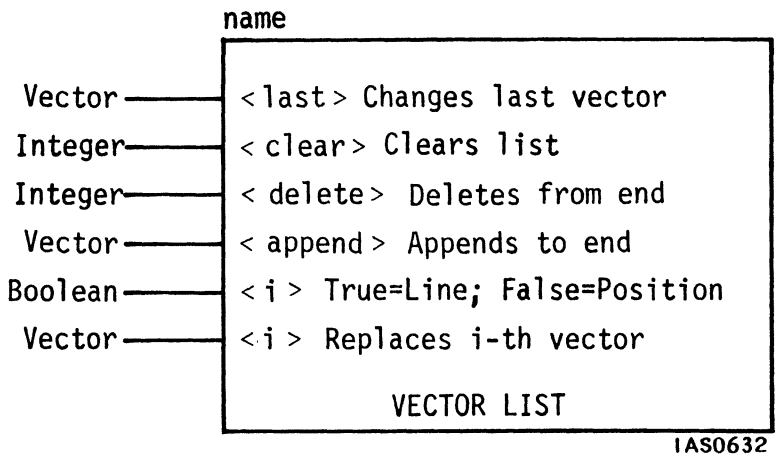
TEXT SIZE



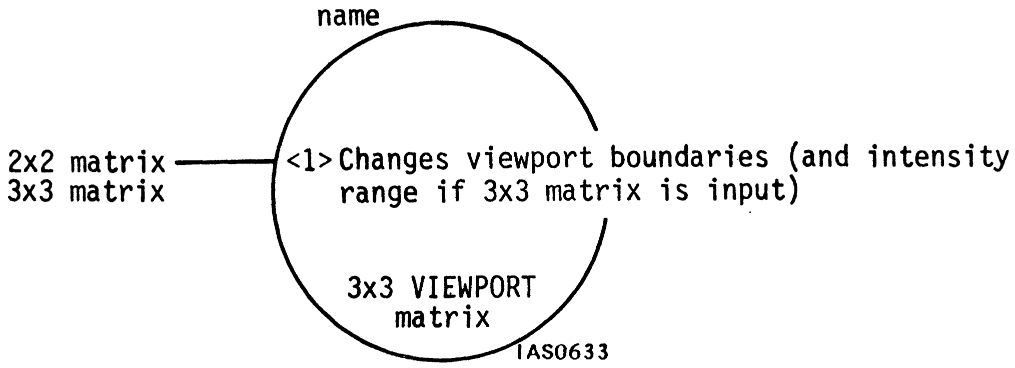
TRANSLATE



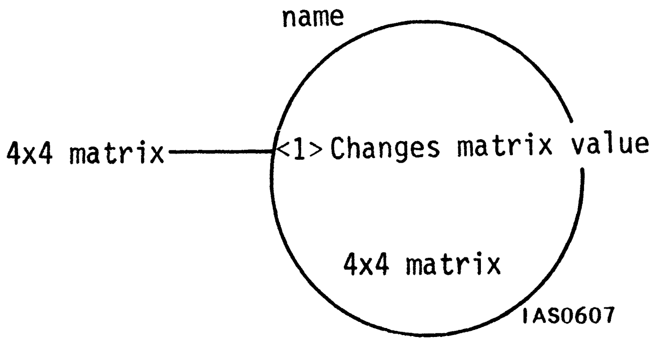
VECTOR_LIST


name
Vector
Integer
Integer
Vector
Boolean
Vector
< last > Changes last vector
< clear > Clears list
< delete > Deletes from end
< append > Appends to end
< i > True=Line; False=Position
< i > Replaces i-th vector
VECTOR LIST
IAS0632

VIEWPORT


name
2x2 matrix
3x3 matrix
<1> Changes viewport boundaries (and intensity range if 3x3 matrix is input)
3x3 VIEWPORT matrix
IAS0633

WINDOW



# ASCII Character Code Set

| Decimal Value | ASCII Character | Decimal Value | ASCII Character | Decimal Value | ASCII Character |
|---|---|---|---|---|---|
| 0 | NUL | 44 | , | 88 | X |
| 1 | SOH | 45 | - | 89 | Y |
| 2 | STX | 46 | . | 90 | Z |
| 3 | ETX | 47 | / | 91 | [ |
| 4 | EOT | 48 | 0 | 92 | \ |
| 5 | ENQ | 49 | 1 | 93 | ] |
| 6 | ACK | 50 | 2 | 94 | ↑ or ^ |
| 7 | BEL | 51 | 3 | 95 | ← or _ |
| 8 | BS | 52 | 4 | 96 | ` |
| 9 | HT | 53 | 5 | 97 | a |
| 10 | LF | 54 | 6 | 98 | b |
| 11 | VT | 55 | 7 | 99 | c |
| 12 | FF | 56 | 8 | 100 | d |
| 13 | CR | 57 | 9 | 101 | e |
| 14 | SO | 58 | : | 102 | f |
| 15 | SI | 59 | ; | 103 | g |
| 16 | DLE | 60 | < | 104 | h |
| 17 | DC1 | 61 | = | 105 | i |
| 18 | DC2 | 62 | > | 106 | j |
| 19 | DC3 | 63 | ? | 107 | k |
| 20 | DC4 | 64 | @ | 108 | l |
| 21 | NAK | 65 | A | 109 | m |
| 22 | SYN | 66 | B | 110 | n |
| 23 | ETB | 67 | C | 111 | o |
| 24 | CAN | 68 | D | 112 | p |
| 25 | EM | 69 | E | 113 | q |
| 26 | SUB | 70 | F | 114 | r |
| 27 | ESC or ALT | 71 | G | 115 | s |
| 28 | FS | 72 | H | 116 | t |
| 29 | GS | 73 | I | 117 | u |
| 30 | RS | 74 | J | 118 | v |
| 31 | VS | 75 | K | 119 | w |
| 32 | SP | 76 | L | 120 | x |
| 33 | ! | 77 | M | 121 | y |
| 34 | " | 78 | N | 122 | z |
| 35 | # | 79 | O | 123 | { |
| 36 | $ | 80 | P | 124 | \| |
| 37 | % | 81 | Q | 125 | } |
| 38 | & | 82 | R | 126 | ~ Tilde |
| 39 | ' | 83 | S | 127 | Rubout or DEL |
| 40 | ( | 84 | T | | |
| 41 | ) | 85 | U | | |
| 42 | * | 86 | V | | |
| 43 | + | 87 | W | | |

# PS 300 GRAPHICS FIRMWARE RELEASE NOTES

**Version A2.V01**
**(904015-602)**

**June, 1986**

Version A2.V01 of the PS 300 Graphics Firmware supersedes all previous releases and is the only firmware version now supported by E&S Customer Engineering. These Release Notes summarize changes and additions to the Graphics Firmware and are intended for use with the entire PS 300 family of graphics computers. Information specific to a particular model is noted.

Formal change pages for the Command and Function Summaries in the PS 300 Document Set are provided with this release. Please discard the old pages and replace with these new pages.

Before you use the new firmware, read these Release Notes carefully and be sure you understand the differences between this and previous releases.

With this release, PS 300 Diagnostic Diskettes are no longer supplied. Instead, one Diagnostic Utility Diskette is provided containing all the utility programs described in Volume 5, Section 10 of the PS 300 Document Set. Please refer to that section for instructions on using the utility programs for back-up and file management and make note that the new Diagnostic Utility Diskette is the only diskette that should be used to load these programs.

Direct your questions and comments to the Evans & Sutherland's Customer Engineering Hotline 1-800-582-4375 (except Utah). Within Utah, customers should call 582-5847.

## This Release Package Includes the Following Items

- One copy of the Graphics Firmware Version A2.V01.
- A magnetic distribution tape including (but not limited to) the following:
  - An updated version of the PS 300 Graphics Support Routines on magnetic tape. The files READFOR.GSR and READPAS.GSR contain descriptions of the FORTRAN and Pascal GSR software.
  - The PS 300 Host-Resident I/O Subroutines
  - Three programming utilities: NETEDIT, NETPROBE, and MAKEFONT (For VAX/VMS users only).
- One copy of the Diagnostic Utility Diskette.
- These Release Notes, summarizing the new features of the A2.V01 release and listing corrected problems, miscellaneous notes, and advice. These notes should be placed in the new PS 300 Document Set behind the Release Notes tab in Volume 3A.
- One copy of Graphics Firmware Version A1.V03 for single diskette systems. These systems do not support the Writeback feature.
- Writeback Feature User's Guide, detailing the new Writeback feature available with this release.

## New Distribution Tape Format

All PS 300 VAX/VMS sites will receive the A2.V01 distribution tape (PS 300 host software) in VMS Backup format. To install the VAX PS 300 host software, first create a subdirectory for the PS 300 software and set your default to that directory by following the procedure below. Using the VMS Backup Utility, enter the following commands:

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

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

This will create the sub-directory A2V01.DIR which is the parent directory of the PS 300 host software.

All PS 300 sites that are not DEC VAX/VMS, excluding UNIX and IBM sites, will receive a variable length ANSI format distribution tape with the PS 300 host software. Consult your system operation manual for instructions on reading ANSI-formatted tapes.

All UNIX and IBM sites will receive the distribution tape with the same format as previous releases.

## Enhancements in Graphics Firmware Version A2.V01

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

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

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

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

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

## Modifications in the Graphics Firmware

- Changes to BUTTONSIN (PS 350 Only)

  The initial function instance BUTTONSIN has two new inputs.

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

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

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

Most Significant Bit

Least Significant Bit

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

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

- Changes to ONBUTTONLIGHTS and OFFBUTTONLIGHTS (PS 350 Only)

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

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

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

## PS 300 Bug Fixes

- Begin_Structure Name

  Within Begin_Structure, End_Structure pairs, names are concatenated with the name of the most recent Begin_Structure. Thus, in the following example, the names (with the outermost Begin_Structure) should be name.name1, name.name2, name.name3, and name.name4.

```
Name := begin_s
   begin_s
       name1 := ...
       name2 := ...
   end_s;
   name3 := ...
   begin_s
       name4 := ...
   end_s;
end_s;
```

  In previous releases, the names were name.name1, name.name2, name3, name4. This occurred because the first End_Structure after the unnamed Begin_Structure removed the outer name, and so all names afterward did not have the prefix. This bug was fixed so that all names are concatenated correctly. Some users may notice this if they have written their code to get around the bug.

- Wildcard Delete

  The wildcard delete command has been modified so that only the named entities created by the command interpreter that receives the wildcard delete command are deleted. This change fixes the problem of accidentally deleting system named entities such as CURSOR and HOST_MESSAGE.

- XFORMDATA Vector Loss

  A bug has been fixed in the XFORMDATA function that caused certain dots in large dots vector lists to be missing from the vector list collected by XFORMDATA.

- User-Written Function SRECORDS

  User-Written Function SRECORDS can now be correctly loaded using the GSRs on all supported interfaces. In the A1.V01 distribution, SRECORDS could not be loaded using the GSR's on the DMR11 and PARALLEL interfaces. In the A1.V02 distribution user-written function SRECORDS could be loaded using the GSRs but a problem in the CONFIG.DAT file caused subsequent GSR data to be loaded incorrectly. This problem has been fixed.

# PS 300 WRITEBACK FEATURE

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

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

This guide contains:

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

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

## Writeback User Interface

The Writeback feature is implemented by:

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

## WRITEBACK Operation Node

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

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

```
Name := WRITEBACK [APPlied to Name1];
```

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

```
SEND TRUE TO <1>name;
```

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

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

**NOTE**

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

## WRITEBACK Operation Node Constraints

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

**E 8 ACP cannot find your operate node**

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

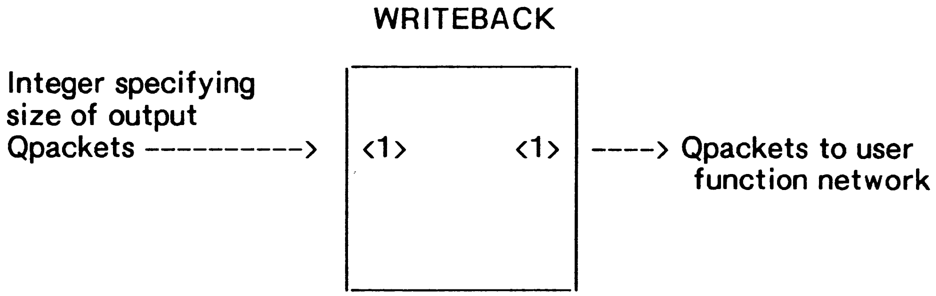
The terminal emulator and message_display information will not be returned to the host.

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

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

## The WRITEBACK Function

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


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

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

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

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

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

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

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

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

## Data Output by WRITEBACK

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

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

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

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

### Data Packets Returned

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

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

```
bits 15|14                    0|
     |0| command               |
     |                         |
     |-------------------------|
     |    parameter            |
     |_________________________|
```

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

```
bits    15|14|13|12 -- 6 |5 --    0|
         1| d|//|   inten|//////// |   if d = 1, then it is a DRAW
                                       if d = 0, it is a MOVE
bits    15 - 13 |12   --          0|
        ////////|      y coord     |

bits    15 - 13 |12   --          0|
        ////////|      x coord     |
```

**NOTE**

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

### Command Descriptions

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

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

START-OF-WRITEBACK

code in hex = H#0B00
# 2816

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

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

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

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

END-OF-WRITEBACK

code in hex = H#0C00
# 3072

Parameters:
None

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

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

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

LINE GENERATOR STATUS

code in hex = H#0200
# 512

Parameters:
Status word (one word)

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

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

Line Generator Status Register (LGS):

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

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

COLOR

code in hex = H#0400
# 1024

Parameters:
Color value (one word)

| | |
|---|---|
| 400 | |
| Hue | Saturation |
| ///////////////////// | |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| /// | | | | | | | | /// | | | | | /////////// | | |
| /// | HI | | | HUE | | | LO | //// | HI | SAT | | LO | /////////// | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |

TEXTURE

code in hex = H#0500
# 1280

Parameters:
Texture value (one word)

| | |
|---|---|
| 500 | |
| ///////// | Texture |
| //////////////////// | |

Line Generator Texture Register:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ///////////////////////////////// | | | | | | | | Texture bit pattern | | | | | | | |
| ///////////////////////////////// | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |

H#007F or H#00FF both default to a Solid line.
For non-PS 350 users, the texture will always be H#00FF.

The following commands are for PS 350 users ONLY.

CLEAR DDA — code in hex = H#0100 — # 256

Parameters:
None

PICK BOUNDARY — code in hex = H#0300 — # 768

Parameters:
Four Boundary Values (4 words)

CLEAR SAVE POINT — code in hex = H#0600 — # 1536

Parameters:
None

SET PICK ID — code in hex = H#0700 — # 1792

Parameters:
Pick ID Pointer (two words)

SET LightPen MODE — code in hex = H#0800 — # 2048

Parameters:
Control Mask (1)
New X,Y (2)
Delta distance (1)
Delta frames (1) (Total five words)

ENABLE PICK — code in hex = H#0900 — # 2304

Parameters:
None

DISABLE PICK — code in hex = H#0A00 — # 2560

Parameters:
None

SET BLINK RATE — code in hex = H#0D00
# 3328

Parameters:
Blink Rate (one word)

ENABLE BLINK — code in hex = H#0E00
# 3584

Parameters:
None

DISABLE BLINK — code in hex = H#0F00
# 3840

Parameters:
None

END-OF-FRAME — code in hex = H#1700
# 5888

Parameters:
None

Signifies that the current update cycle is completed and that any following data is part of the next update frame. This also signifies end of the writeback segment.

VIEWPORT CENTER — code in hex = H#1800

Parameters:
x center (one word)
y center (one word)
z center (one word)
spare (two words)

```
  bits 15 ..................... 0
      | coordinates              |  2's complement vector
      |__________________________|
```

This value has to be added to each x,y coordinate pair. This information is necessary to calculate the actual coordinates of the data which has been viewport scaled. Every time a new viewport is traversed by the Arithmetic Control Processor, a new viewport center command will be sent.

**NOTE**

Codes H#1900 - H#1F00 are reserved for future commands. Code H#0000 is defined as a no-op, and naturally has no parameters.

## EXAMPLE OF THE SEQUENCE OF DATA SENT BACK TO THE HOST

The following example illustrates the sequence of data and the data in byte format sent to the host during a WRITEBACK operation.

| | | |
|---|---|---|
| B00 | | Start-of-writeback command |
| ////////// | Texture | |
| LGS | | |
| 400 | | Color command |
| Hue | Saturation | |
| //////////////////////// | | |
| Intensity | | V |
| Y | | E |
| X | | C |
| : | | T |
| : | | O |
| : | | R |
| : | | S |
| : | | |
| : | | |
| : | | |
| : | | |
| 200 | | Line Generator Status command |
| LGS | | |
| //////////////////////// | | |
| 500 | | Texture command |
| ////////// | Texture | |
| //////////////////////// | | |
| 400 | | Color command |
| Hue | Saturation | |
| //////////////////////// | | |
| Intensity | | V |
| Y | | E |
| X | | C |
| : | | T |
| : | | O |
| : | | R |
| : | | S |
| : | | |
| : | | |
| : | | |
| : | | |
| : | | |
| C00 | | End-of-writeback command |
| 0/1 | | 0 = finished successfully, 1 = cannot |
| //////////////////////// | | finish because of insufficient memory |

## Data in Byte Format

```
0B  00    Start-of-writeback command
00  FF    Texture
04  70    LGS
04  00    Color command
80  00    Hue/Saturation
00  00    Not used
00  FF    Intensity
1Y  FF      Y
1X  FF      X
00  FF    Intensity
2Y  FF      Y
2X  FF      X
 :          :
 :          :
 :          :
02  00    LGS command
04  70    LGS
00  00    Not used
05  00    Texture command
00  FF    Texture
00  00    Not used
04  00    Color command
80  00    Color
00  00    Not used
00  FF    Intensity
1Y  FF      Y
1X  FF      X
 :          :
 :          :
 :          :
0C  00    End-of-writeback command
00  00    Finshed successfully
00  00    Not used.
```

## SAMPLE WRITEBACK PROGRAM

```
PROGRAM Writeback(Input,Output,Outfile,Devfile);
{ Program to read writeback data from a PS 350. This program sets up a  }
{ function network to get the writeback data and processes the data and }
{ creates a data file on the host with the data from the PS 350.        }

CONST
 %INCLUDE 'PROCONST.PAS'
 Max_buf = 1024;

TYPE
 Int16 = -32768..32767;
 Max_line = VARYING [Max_buf] OF CHAR;
 %INCLUDE 'PROTYPES.PAS'

VAR
 OUTFILE : TEXT;
 DEVFILE : TEXT;
 DEVSPEC : P_VARYINGTYPE;
 OUTNAME : P_VARYINGTYPE;
 WBNAME  : P_VARYINGTYPE;
 COMMAND : INT16;
 INDEX : INTEGER;
 LEN : INTEGER;
 Inline : P_VARYBUFTYPE;
 vx,vy,vz : REAL;
 In_DDA : BOOLEAN := FALSE;

 %INCLUDE 'PROEXTRN.PAS'

  PROCEDURE ERR (ERROR: INTEGER);
  {}
  { ERROR HANDLER ROUTINE }
  {}
    BEGIN { ERR }
      {}
      WRITELN(' ERROR :=',ERROR);
      HALT;
      {}
    END; { ERR }
  PROCEDURE Setup;
  { Create function network to send writeback data to host }
  { This uses F:cvt8to6 to send 6-bit data to the host }
      BEGIN
      PFnInst('cvt','cvt8',Err);
      Pconnect ('Writeback',1,1,'cvt',Err);
      Pconnect ('cvt',1,1,'host_message', Err);
      PsndStr (CHR(36),2,'cvt',Err);
      PsndFix (48,1,'writeback', Err);
      PNameNil('screensave',Err);
      PPurge( Err);
      END;

{ Utility procedures}
  PROCEDURE Six_to_eight( Inbuf :  Max_line;
   VAR Outbuf : P_VARYBUFTYPE);
  { Data from PS 350 is in six-bit packed format. This procedure unpacks
    data}

  CONST Base = 36;

  TYPE
    Cheat_4 = PACKED RECORD CASE Boolean OF
   TRUE : ( i: UNSIGNED);
   FALSE : ( c: PACKED ARRAY [1..4] OF CHAR);
 END;

  VAR
    w : Cheat_4;
    c_out,cycle_count,buf_index,il,tc : INTEGER;
    first : BOOLEAN;

  BEGIN
    buf_index := 1;
    first := TRUE;
    cycle_count := 1;
    c_out := 4;
    outbuf := '';
    WHILE buf_index <= len DO
      BEGIN
 tc := ORD(Inbuf[buf_index]) - base;
 IF first THEN
   IF tc < 0 THEN
     c_out := 4+tc
   ELSE
     BEGIN
       first := FALSE;
       w.i := tc;
       cycle_count := SUCC(cycle_count);
     END { ELSE tc >= 0 }
```

```
ELSE
  BEGIN
    w.i := w.i * (2**6);
    w.i := UOR(w.i ,tc);
    cycle_count := SUCC(cycle_count);
  END; { ELSE }
IF cycle_count > 6 THEN
  BEGIN
    FOR il := 4 DOWNTO (5-c_out) DO
      Outbuf := outbuf + w.c[il];
    cycle_count := 1;
    first := true;
  END;
buf_index := SUCC(buf_index);
     END; { WHILE }
 END;

 PROCEDURE Next_Block;
 { Get a block of data from the PS 350 and convert from six to eight}
 { bit format }

 VAR Inbuff : Max_line;

 BEGIN
   PGETWAIT(Inbuff,err);
   Index := 1;
   Len := LENGTH(Inbuff);
   Six_to_eight ( Inbuff, Inline);
   Len := LENGTH(Inline);
 END;

 PROCEDURE Get_Value( VAR a : INT16);
 { Convert two bytes of input buffer to 16 bit integer }

 VAR i : INTEGER;

 BEGIN { Get_Value }
   a := 0;
   FOR i := 1 TO 2 DO
     BEGIN
Index := Index + 1;
IF Index > Len THEN
  Next_Block;
a := a * 256 + ORD(Inline[Index]);
     END;
 END;{ Get_Value }
```

```
{ Procedures for processing refresh buffer commands }

  PROCEDURE Clear_DDA;
  { CLEAR DDA - %X0100 }
  { Parameters - None }
  { Indicates start of sequence to set viewport center }
  { This sequence is CLEAR DDA, CLEAR SAVE POINT, Vector, CLEAR SAVE POINT}

  VAR a,b : Int16;

  BEGIN
    In_DDA := TRUE;
    Get_value ( a );
    Get_value ( b );
    Writeln(Outfile,'{Clear DDA}');
  END;

  PROCEDURE Write_LGS;
  { WRITE LINE GENERATOR STATUS - %X0200 }
  { Parameters - Status word (one word)  }
  { Bit   8 : Dot mode.  }
  { Bit   6 : Fast sweep ( Opposite of 7) }
  { Bits  5 -  4: Contrast selection (00-min,11-max)}
  { Bits  3 -  0: Scope select( 1 disables,0 enables)}

  VAR lgs,a : Int16;

  BEGIN
    Get_value ( lgs );
    Get_value ( a );
    Writeln(Outfile,'{Write LGS:',HEX(lgs),'}');
  END;

  PROCEDURE Write_Pick_Bound;
  { WRITE PICK BOUNDARY - %X0300  }
  { Parameters - Left, Right, Bottom, Top }

  VAR l,r,b,t,a : Int16;

  BEGIN
    Get_value ( l );
    Get_value ( r );
    Get_value ( b );
    Get_value ( t );
    Get_value ( a );
    Writeln(Outfile,'{Write_Pick_bound:',HEX(l),HEX(r),HEX(b),HEX(t),'}');
  END;
```

```
PROCEDURE Write_Color;
{ WRITE COLOR - %X0400  }
{ Parameters - Color value (one Word) }
{ Bit  15 : Not Used  }
{ Bits 14 - 8 : Hue (High order in 14)}
{ Bit   7 : Not Used  }
{ Bits  6 - 3 : Sat (High order in 3) }
{ Bits  2 - 0 : Not Used  }

VAR c,a : Int16;

BEGIN
  Get_value ( c );
  Get_value ( a );
  Writeln(Outfile,'{Write_Color:',HEX(c),'}');
END;

PROCEDURE Write_Texture;
{ WRITE TEXTURE - %X0500      }
{ Parameters - Texture value (one word)   }
{ Bits 15 - 7 : Not Used      }
{ Bits  6 - 0 : Texture bit pattern     }

VAR t,a : Int16;

BEGIN
  Get_value ( t );
  Get_value ( a );
  Writeln(Outfile,'{Write_Texture:',HEX(t),'}');
END;

PROCEDURE Clear_Save_Point;
{ CLEAR SAVE POINT - %X0600 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Clear_Save_Point:}');
END;

PROCEDURE Set_Pick_Id;
{ SET PICK ID - %X0700      }
{ Parameters - Pick Id Pointer (two words)}

VAR a,b : Int16;
```

```
BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Set_Pick_Id:',HEX(a),HEX(b),'}');
END;

PROCEDURE Set_Lightpen_Mode;
{ SET LIGHTPEN MODE - %X0800 }
{ Parameters - Control mask     }
{   Tracking cross y   }
{   Tracking cross x   }
{   Delta distance     }
{   Delta frames     }

VAR cm,x,y,dd,df : Int16;

BEGIN
  Get_value ( cm );
  Get_value ( x );
  Get_value ( y );
  Get_value ( dd );
  Get_value ( df );
  Writeln(Outfile,'{Set_Lightpen_mode:',HEX(cm),HEX(x),HEX(y),
    HEX(dd),HEX(df),'}');
END;

PROCEDURE Enable_Pick;
{ ENABLE PICK - %X0900}
{ Parameters - None }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Pick:}');
END;

PROCEDURE Disable_Pick;
{ DISABLE PICK - %X0A00   }
{ Parameters - None     }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Pick:}');
END;
```

```
PROCEDURE Enable_Writeback;
{ ENABLE WRITEBACK - %X0B00 }
{ Parameters - Line Texture }
{   Line Gen Status}

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Writeback:',HEX(a),HEX(b),'}');
END;

PROCEDURE Disable_Writeback;
{ DISABLE WRITEBACK - %X0C00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Writeback:}');
END;

PROCEDURE Set_Blink_Rate;
{ SET BLINK RATE - %X0D00 }
{ Parameters - Blink rate }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Set_Blink_Rate:',HEX(a),'}');
END;

PROCEDURE Enable_Blink;
{ ENABLE BLINK - %X0E00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Blink:}');
END;
```

```
PROCEDURE Disable_Blink;
{ DISABLE BLINK - %X0F00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Blink:}');
END;

PROCEDURE End_Of_Frame;
{ END OF FRAME - %X1700    }
{ Parameters - None     }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{End_Of_Frame:}');
END;

PROCEDURE Viewport_Center;
{ VIEWPORT CENTER - %X1800}
{ Parameters - x center    }
{   y center    }
{   z center    }

VAR xc,yc,zc,a,b : Int16;

BEGIN
  Get_value ( xc );
  Get_value ( yc );
  Get_value ( zc );
  Get_value ( a );
  Get_value ( b );
  vx := xc;
  IF (vx >= 32768) THEN vx := vx - 65536.0;
  vx := vx/32767;
  vy := yc;
  IF (vy >= 32768) THEN vy := vy - 65536.0;
  vy := vy/32767;
  vz := zc;
  IF (vz >= 32768) THEN vz := vz - 65536.0;
  vz := vz/32767;
  Writeln(Outfile,'{Viewport_Center:',vx:6:6,' ',vy:6:6,' ',vz:6:6,'}');
END;
```

```
PROCEDURE Process_Vector;
{ Vector - Bit 15 of command = 1 }
{ Word 1 ( command )   }
{ Bit  15 : Always one for vector }
{ Bit  14 : 1 = Draw, 0 = Move }
{ Bits 12 - 6 : Intensity/2  }
{ Bits  5 - 0 : Not Used  }
{ Word 2 ( y coord)   }
{ Bits 15 - 13: Not Used  }
{ Bits 12 -  0: Y coordinate  }
{ Word 3 ( x coord)   }
{ Bits 15 - 13: Not Used  }
{ Bits 12 -  0: X coordinate  }

VAR a,b : Int16;
  un : UNSIGNED;
  pl : CHAR;
  int,x,y : REAL;

BEGIN
  Get_value ( a );
  Get_value ( b );
  un:=command;
  pl:='l';
  IF (UAND(un,%X4000) = 0) THEN pl := 'p';
  un := UAND(un,%X1FC0);
  int := un;
  IF In_DDA THEN
    vz := int/8128.0
  ELSE
    int := (int/8128.0 + vz) * 2;
  un := a;
  un := UAND(un,%X1FFF);
  y := un;
  IF (y >= %X1000) THEN y := y - %X2000;
  IF In_DDA THEN
    vy := y / %XFFF
  ELSE
    y := y / %XFFF + vy;
  un := b;
  un := UAND(un,%X1FFF);
  x := un;
  IF (x >= %X1000) THEN x := x - %X2000;
  IF In_DDA THEN
    vx := x / %XFFF
  ELSE
    x := x / %XFFF + vx;
  IF In_DDA THEN
    BEGIN
  Writeln(Outfile,'{New View Center:',vx:6:6,' ',vy:6:6,' ',vz:6:6,'}');
  In_DDA := FALSE;
      END
    ELSE
    Writeln(Outfile,'{Vec ',pl,' ',x,',',y,' i=',int,'}');
  END;

  PROCEDURE Unknown;
  VAR a,b : Int16;

  BEGIN
    Get_value ( a );
    Get_value ( b );
    Writeln(Outfile,'{Unknown:',HEX(command),HEX(a),HEX(b),'}');
  END;

BEGIN  { Writeback}
  Write ('Enter Output File Name:');
  Readln(Outname);
  Write ('Enter Writeback Operate Node Name:{WB$ is default mode}');
  Readln(wbname);
  open(Outfile,Outname,new);
  rewrite(Outfile);

  { Look for file specifying line for pattach procedure }
  { Example of record in PSDEV.DAT: }
  { 'logdevnam=tt:/Phydevtyp=async' }
  open(devfile,'psdev',old);
  reset(devfile);
  readln(devfile,devspec);
  close(devfile);

  PATTACH(devspec,err);  { Attach to PS 350 }
  Setup;   { Setup writeback network }

  PNAMENIL('SCREENSAVE', ERR);
  PPURGE(ERR);
  PSndBool(TRUE,1,wbname, Err); { Trigger write back operate }

  Next_block;   { Read in first block of writeback data}

  Index := 0;
  Command := 0;
  vx := 0.0;
  vy := 0.0;
  vz := 0.0;

  { Process writeback buffers until END OF FRAME or END WRITEBACK}
  WHILE (Command <> %X0C00) AND (Command <> %X1700) DO
```

```
    BEGIN
      Get_value(Command);
      IF (Command > 32767) THEN { If bit 15 of command if set}
 Process_vector
   ELSE
   CASE (Command DIV 256) OF
     %X01 : Clear_DDA;
     %X02 : Write_LGS;
     %X03 : Write_Pick_Bound;
     %X04 : Write_Color;
     %X05 : Write_Texture;
     %X06 : Clear_Save_Point;
     %X07 : Set_Pick_Id;
     %X08 : Set_Lightpen_Mode;
     %X09 : Enable_Pick;
     %X0A : Disable_Pick;
     %X0B : Enable_Writeback;
     %X0C : Disable_Writeback;
     %X0D : Set_Blink_Rate;
     %X0E : Enable_Blink;
     %X0F : Disable_Blink;
     %X17 : End_Of_Frame;
     %X18 : Viewport_Center;
     OTHERWISE Unknown;
   END; { CASE }
    END;
  PFNINST('SCREENSAVE', 'SCREENSAVE', ERR  PDETACH(ERR);
  PPURGE(ERR):
  {}
END.  { Writeback}
```

# CHANGE PAGES FOR THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND THE GRAPHICS SUPPORT ROUTINE MANUALS

## FORMAT

```
name := RAWBLOCK i;
```

## DESCRIPTION

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

## PARAMETERS

i – bytes available for use.

## NOTES

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

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

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

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

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

## DISPLAY TREE NODE CREATED

Rawblock data node.

## FORMAT

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

## DESCRIPTION

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

## PARAMETERS

**name** – Any legal PS 300 name.

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

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

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

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

3. Connectivity:

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

PARAMETERS (continued)

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

C. **DOTs** - Each vector specifies a dot.

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

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

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

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

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

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

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

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

n - Estimated number of vectors.

PARAMETERS (continued)

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

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

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

For **ITEMized** vector lists the syntax is:

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

or

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

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

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

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

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

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

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

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

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

## DEFAULTS

If not specified, the options default to:

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

Non color-blended vectors default to:

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

If i is not specified, it defaults to 1.

Color-blended vectors default to:

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

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

Tabulated vectors default to:

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

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

## NOTES

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

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

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

NOTES (continued)

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

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

6. Options must be specified in the order given.

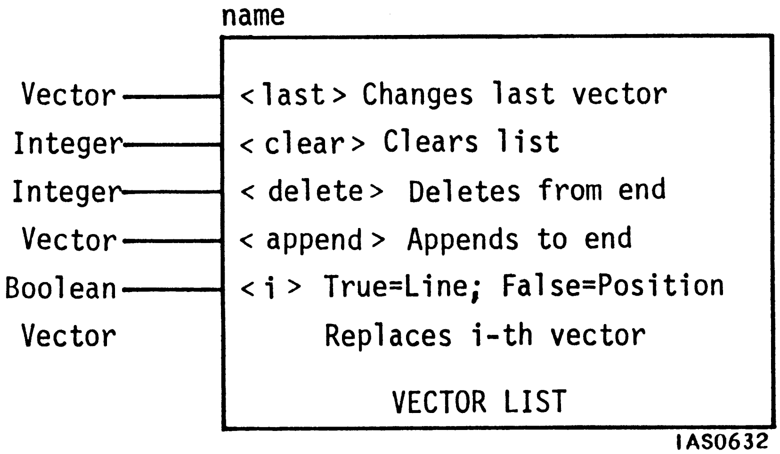
7. If CONNECTED_lines, SEParate_lines, or DOTs are specified in the options area but the vectors are entered using P/Ls, then the option specified takes precedence.

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

## DISPLAY TREE NODE CREATED

Vector list data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

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

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

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

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

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

EXAMPLES

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

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

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

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

Version A2.V01

## FORMAT

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

## DESCRIPTION

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

## PARAMETERS

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

## NOTES

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

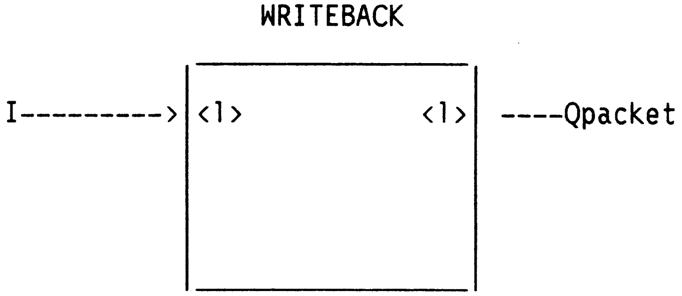
2. Only a structure that is being displayed can be enabled for writeback. This means that the WRITEBACK operation node must be traversed by the display processor and so must be included in the displayed portion of the structure. If the writeback of only a portion of the picture is desired, WRITEBACK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

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

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

## DISPLAY TREE NODE CREATED

The command creates a WRITEBACK operation node.



## PURPOSE

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

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

## DESCRIPTION

### INPUT

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

### OUTPUT

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

## NOTES

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

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

Name := WRITEBACK

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

## PARAMETERS

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

## PS 300 COMMAND AND SYNTAX

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

Name := WRITEBACK

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

## PARAMETERS

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

## PS 300 COMMAND AND SYNTAX

name := WRITEBACK [APPLied to Name1];

Name := WRITEBACK

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

where:

Namel is a CHARACTER STRING
Errhnd is the user-defined error-handling subroutine
```

## DEFINITION

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

## PARAMETERS

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

## PS 300 COMMAND AND SYNTAX

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

## UTILITY PROCEDURE AND PARAMETERS

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

## DEFINITION

This procedure attaches the PS 300 to the communications channel.

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

The parameter (Modify) must contain the phrases:

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

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

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

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

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

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

PVecBegn

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

PVecEnd

It contains the following parametric definitions:

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

(Continued on next page)

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

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

NOTE

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

PVecBegn

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

PVecEnd

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

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

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

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

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

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

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

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

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

Together, the above 4 procedures implement the PS 300 command

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

## UTILITY SUBROUTINE AND PARAMETERS

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

where:

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

## DESCRIPTION

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

The parameter (Modify) must contain the phrases:

LOGDEVNAM=name/PHYDEVTYP=type

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

ASYNC (standard RS-232 asynchronous communication interface)
DMR-11 (high-speed synchronous interface)
PARALLEL (high speed parallel interface
ETHERNET (DECnet Ethernet option).

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

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

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

(Continued on next page)

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

where:

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

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

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

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

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

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

ErrHnd is the user-defined error-handler subroutine.

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

PVcBeg

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

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

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

### NOTE

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

(Continued on next page)

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

where:

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

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

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

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

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

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

ErrHnd is the user-defined error-handler subroutine.

## DESCRIPTION

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

PVcBeg

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

PVcEnd

(Continued on next page)

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

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

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

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

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

## DEFINITION

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

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

Together, the above 4 procedures implement the PS 300 command

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

PVecBegn

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

PVecEnd

It contains the following parametric definitions:

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

(Continued on next page)

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

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

NOTE

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

PVecBegn

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

PVecEnd

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

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

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

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

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

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

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

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DEFINITION

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

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

Together, the above 4 procedures implement the PS 300 command

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

where:

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

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

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

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

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

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

ErrHnd is the user-defined error-handler subroutine.

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

PVcBeg

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

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

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

### NOTE

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

(Continued on next page)

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

where:

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

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

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

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

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

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

ErrHnd is the user-defined error-handler subroutine.

## DESCRIPTION

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

PVcBeg

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

PVcEnd

(Continued on next page)

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

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

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

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

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

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

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

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

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

## DEFINITION

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

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

Together, the above 4 procedures implement the PS 300 command

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

Table 1. Key to Abbreviations for Valid Data Types

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

## Conjunctive/Disjunctive Sets

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

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

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

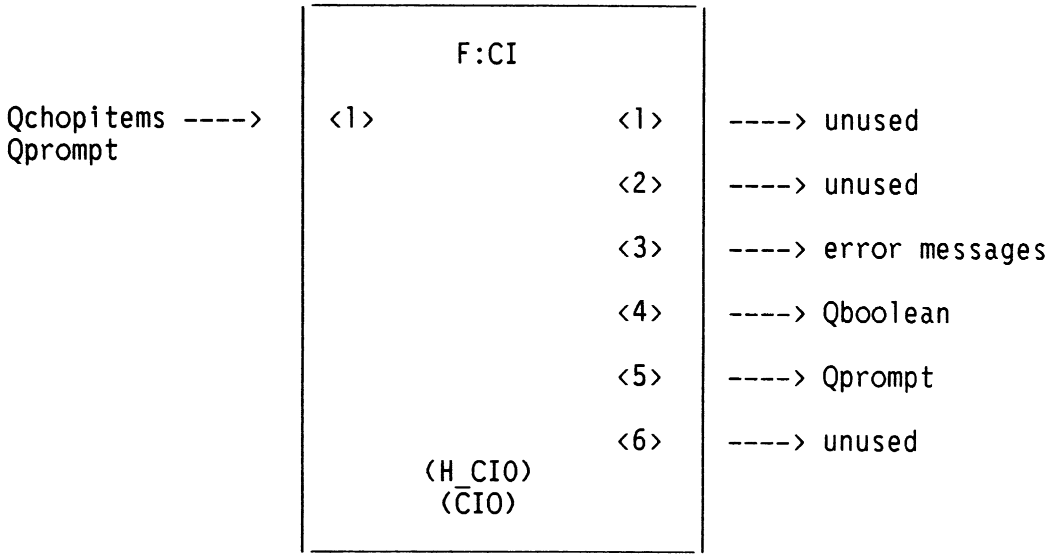
F:BROUTE, on the other hand, is a conjunctive-input, disjunctive-output function. Both inputs require messages to activate the function. However, a message will be sent out only one of the outputs, depending on the value received on input 1.

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

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

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

F:CI



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

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

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

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

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

### 7.5.3 Command Status Command

The command:

```
COMMAND STATUS;
```

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

### 7.5.4 Reboot Command

The command

```
   REBOOT password;
```

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

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

### 7.5.5 Set Priority

The command

```
   Set Priority of name to i;
```

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

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

### 7.5.6 Notes On Using the CONFIGURE Mode

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

TABLE 9-1 PS 300 TRAPS and Their Meanings

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

# PS 340 GRAPHICS FIRMWARE RELEASE NOTES

**Version A2.V01**

**June, 1986**

## 1. GENERAL INTRODUCTION

Version A2.V01 of the PS 340 Graphics Firmware supersedes all previous releases and is the only firmware version now supported by E&S Customer Engineering. These Release Notes summarize the changes and additions to the Graphics Firmware and are intended for use with all PS 340 systems supporting calligraphic and/or raster displays.

There are two versions of this PS 340 release, each distributed on a separate diskette. Both diskettes are included in this release package.

Version A2.V01 PS 340S incorporates new features and changes to previous releases. This version supports hidden-line rendering and shading operations and runs on systems with a 2K ACP.

Version A2.V01 PS 340E incorporates all new features and changes to previous releases (same as above) with additional enhancements and capabilities. This version is supported only on systems with a 4K ACP.

The PS 340E (4K ACP) firmware will boot on a PS 340 with a 2K ACP, but the system will be unable to perform any rendering and other operations.

You can determine whether your system has a 2K or a 4K ACP by booting the system using the PS 340E (4K ACP) firmware and entering the following commands:

```
Reserve 100000;
X := Surface then y;
Y := Polygon -1,1,0  1,1,0  1,-1,0  -1,-1,0;
Display X;
Send fix(3) to <1>x;
```

If the PS 340 vector screen goes blank, the system has a 2K ACP. The screen goes blank because the ACP has referenced control store that does not exist and has stopped traversing data structures.

These Release Notes summarize the changes and additions to the PS 340 Graphics Firmware and are organized as follows. Section 3 discusses modifications in the PS 340 Graphics Firmware that are incorporated in both PS 340S and PS 340E versions (2K ACP and 4K ACP versions). All PS 340 users should read this section. Section 4 gives information on enhancements and new capabilities in the PS 340 that are incorporated only in version PS 340E (4K ACP users only). Section 5 gives programming examples and diagrams.

Formal change pages for the Command and Function Summaries in the PS 300 Document Set are provided with this release. Please discard the old pages and replace with these new pages.

Before you use the new firmware, read these Release Notes carefully and be sure you understand the differences between this and previous releases. Also read the PS 300 Release notes included with this Release package.

Direct your questions and comments to the Evans & Sutherland's Customer Engineering Hotline 1-800-582-4375 (except Utah). Within Utah, customers should call 582-5847.

## 2. RELEASE PACKAGE CONTENTS

This Release Package contains the following items.

- One copy of the PS 340 Graphics Firmware supporting systems with 2K ACP.
- One copy of the PS 340E Graphics Firmware supporting systems with 4K ACP.
- A magnetic tape with all PS 300 graphics support routines including all updated PS 340 FORTRAN and Pascal Routines.
- One copy of the Diagnostic Utility Diskette.
- Change pages for the PS 300 Command and Function Summaries supporting PS 340 Graphics Firmware, Version A2.V01.
- PS 300 Graphics Firmware Release Notes.
- PS 340 Graphics Firmware Release Notes.

## 3. MODIFICATIONS IN THE PS 340 GRAPHICS FIRMWARE

The following changes to the previous A1.V02 release are incorporated in both the PS 340S and PS 340E versions of the firmware.

### 3.1 New Shading Algorithm

In the previous PS 340 release, when you requested a shaded image to be displayed on the raster screen, a hidden line rendering of the object was displayed on the calligraphic screen at the same time. This was because the same algorithm was used to compute both hidden-line and shaded-image renderings.

In the A2.V01 release, the algorithm used to produce shaded images on the raster screen is a raster-oriented algorithm which is completely separate from the hidden-line algorithm used to produce hidden-line pictures on the calligraphic screen.

After requesting a shaded image on the raster display (sending a fix(5), fix(6), fix(7), or fix(8) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node), the calligraphic screen will go blank while the image is being painted on the raster display. After the shaded image has been completely painted on the raster screen, the original object (rather than a hidden-line object) will be displayed on the calligraphic screen.

### 3.2 New Shading Features

Following are changes to the way shaded images were produced in previous versions of the firmware.

- A type of Gouraud shading has been added as a shading rendering style. This style of shading is obtained by sending a fix(8) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node. Sending a fix(7) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node produces a type of Phong shading as before. Both styles of shading will produce a smooth shading of the object if you supply normals with each vertex of the polygon. The Gouraud shading is faster but does not produce the quality of picture that the Phong shading will produce.

- You can now choose between producing an object with jagged polygon edges at a quick rate, or producing an object with smoother edges at a slower rate. This control is accomplished through input <5> of SHADINGENVIRONMENT:

Sending fix(0) to input <5> of SHADINGENVIRONMENT produces no edge-smoothing and is the fastest shading option.

Sending fix(1) to input <5> of SHADINGENVIRONMENT produces smooth edges but may not smooth along edges of interpenetrating polygons or correctly resolve obscurity between surfaces that are extremely close. This has a speed intermediate between a fix(0) and a fix(2).

Sending fix(2) to input <5> of SHADINGENVIRONMENT produces edge-smoothing and is slower.

## 3.3 Transparent Polygon Attribute

Polygons may now have a transparent attribute. To accomplish this, the form of the ATTRIBUTE command has been changed to:

```
[COLOR h[,s[,i]]] [DIFFUSE d] [SPECULAR s] [OPAQUE t]
```

where t refers to a real number between 0 and 1.

The opaque specifier [t] in the ATTRIBUTE command allows the specification of a transparency level and is input as a real number between 0 and 1, with 1 being fully opaque and 0 being fully transparent.

As t decreases from 1 to 0, more of the color of the obscured object(s) will show through. At t=0, the transparent polygon becomes completely invisible. If no opaque attribute is specified, the default is 1 (fully opaque).

Polygons that are rendered as transparent have no color of their own, but merely filter the color of objects appearing behind them. This is according to to the rule that each of the red, green, and blue components of the object behind is multiplied by the red, green, and blue components of the transparent polygon. This means that a transparent object rendered over a black background will be invisible. This also means that a purely blue transparent object rendered over a purely red object, will make the red object look more black (depending on the value of the Opaque specifier).

There are no specular highlights available on transparent objects.

To show polygon orientation relative to the eye point, the color which is transmitted through the transparent object is darkened according to the z-component of a surface normal. This means that with Phong, Gouraud, and flat shading, as the object bends away from the user, the transmitted color becomes darker.

In Phong shading, the surface normal used is the normal derived by interpolating the normals you supply at each vertex. In flat shading, the normal used is the vector perpendicular to the polygon. In Gouraud shading the degree of light which is transmitted is derived by first calculating the degree of light transmitted at the vertices using the normal you supply and then interpolating between the vertices. In wash shading, no surface normal is used and no lights are used.

To render any objects as transparent, you must at some time prior to rendering send a TRUE to input <11> of SHADINGENVIRONMENT. This is a new input that allows you to turn on (or off) transparency. Sending a TRUE to input <11> will cause only the transparent objects to be rendered transparent. Sending a FALSE to input <11> of SHADINGENVIRONMENT will cause all objects to be rendered as fully opaque, regardless of their attributes.

The node created with the ATTRIBUTES command has two new inputs <4> and <14>. Both inputs accept a real number to update the opaque value of the polygon's attributes, for the two sides of the polygon.

### 3.4 Toggling Between the Rendered and the Original Object

Requesting hidden-line pictures, backface pictures, sectioned pictures or cross-sectioned pictures (sending a fix(1), fix(2), fix(3), or fix(4) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node) operates as in previous releases. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node) works only after requesting hidden-line pictures, backface pictures, sectioned pictures or cross-sectioned pictures.

### 3.5 Default Color Look-up Table

This release of the PS 340 firmware includes a new gamma-corrected color look-up table. (This look-up table compensates for the non-linear response of the phosphors in the raster display.) This new table is the default table which will be loaded at boot time. Some users may have applications dependent on the previous look-up table or wish to use the previous table as the default. If you want to use the previous look-up table, you can do so by renaming a file on Disk B of the PS 340 firmware, while running the PS 300 diagnostic operating system.

The new default table is named "LUT.DAT" The old linear look-up table is named "LINLUT.DAT". To use the old linear look-up table as the default table, boot the diagnostic utility diskette and enter the Utility program. Type the command:

```
"RENAME LINLUT.DAT LUT.DAT"
```

All subsequent boots of the firmware will load the old linear look-up table. The new table can be re-enabled by renaming the file again with the command:

```
"RENAME LUT.DAT LINLUT.DAT"
```

## 3.6 SHADINGENVIRONMENT Function Additions

Inputs <8> through <15> have been added to the SHADINGENVIRONMENT function.

User Abort: Input <8> accepts any message and causes an abort to occur for the current rendering. Sometimes the hidden-line algorithm can take a long time to run to completion. This input allows you to exit before the rendering is complete. This functionality only applies to images on the calligraphic screen. It does not apply to images on the raster screen.

Clear/Overlay Control: Input <9> accepts a Boolean which determines whether the screen is to be cleared with the current background color before the rendering is done, or whether the requested rendering will overlay the object already displayed on the raster screen.

Sending a TRUE to input <9> of SHADINGENVIRONMENT causes the current rendered object to be displayed on top of the previous picture displayed on the the raster screen. Sending a FALSE to input <9> of SHADINGENVIRONMENT causes the screen to be cleared with the current background color before the rendering is done. The default is false.

Opaque (Transparency) Control: Input <11> accepts a Boolean which allows you to turn off (or on) the transparency assigned to the polygon with the OPAQUE clause of the attribute command. To render polygons as transparent, at some time prior to the rendering, you must send a TRUE to input <11>. Sending a FALSE to input <11> will cause all objects to be rendered as fully opaque, regardless of their attributes. The default is false.

Specular Highlight Control: Input <12> accepts a Boolean which allows you to turn off (or on) specular highlights for shading on the raster screen.

Flat, Gouraud, and Phong shading in this new release all use the same shading equation. This means that multiple light sources are processed in each case and that specular highlights are calculated. Specular highlights may appear strange in Gouraud and flat shading, so there is an added option to turn off (or on) specular highlights. Sending FALSE to input <12> of SHADINGENVIRONMENT turns off the specular highlight (without requiring a change in the diffuse contribution to the shading equation). Sending TRUE to input <12> of SHADINGENVIRONMENT turns on the specular highlights. The default is true.

Functionality associated with new inputs <10>, <13>, <14>, and <15> is not supported under 2K ACP systems and so is not discussed until Section 4 of these Release Notes.

## 4. ENHANCEMENTS AND NEW CAPABILITIES IN THE PS 340 GRAPHICS FIRMWARE

The following changes to the A1.V02 release are incorporated only in the PS 340E version of the firmware. PS 340E firmware is supported only by systems with a 4K ACP.

### 4.1 Polygon Color Interpolation

You can now specify color at the vertices of a polygon to provide color interpolation across the polygon. To invoke this option, you must first specify the color in the polygon command. To accomplish this, the vertex definition in the polygon command has been changed to:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

where

C - indicates a color to be assigned to the vertex. This color will be interpolated across the polygon to the other vertices.

h, s, i - are coordinates of the Hue-Saturation-Intensity color system.

Polygons may be solidly colored by specifying a color through the attributes command or the colors may be assigned to the vertices by giving a color with each vertex specified. The color is specified by giving, first, the vertex and then, the color (h, s, i). If just the hue and saturation are given, the intensity will default to 1. If just the hue value is given, the saturation and intensity will default to 1. If no vertex colors are given, the vertex colors will default to those specified in the attribute clause.

Vertex colors must be specified for all vertices of a polygon or for none of them. However, as with normals, some polygons may have color at their vertices while other polygons may not have color at their vertices. This means that it is possible to have some objects in the picture color interpolated, while others are not.

Although color of polygon vertices is specified h, s, i, the colors are linearly interpolated across the vertices in the Red-Green-Blue color system. If colors are not interpolating the way you would like them to be add more vertices to the polygon, or break up large solid volumes into smaller sub-volumes and assign the desired colors to the new vertices in the object.

You can specify color for a polygon with both the ATTRIBUTES command and the color by vertex specification. A new input to the SHADINGENVIRONMENT function allows you to switch between attribute-defined color and vertex-defined color. Input <10> of SHADINGENVIRONMENT accepts a Boolean to determine how color will be specified. To use vertex colors rather than surface attributes, send TRUE to input <10> of SHADINGENVIRONMENT. To return to using the attributes specified in the ATTRIBUTE command, send FALSE to input <10> of SHADINGENVIRONMENT. The default is false.

Refer to Section 5 for a programming example showing how to specify vertex colors to provide color interpolation across a polygon.

## 4.2 SHADINGENVIRONMENT Function Additions

Some of the inputs to the SHADINGENVIRONMENT FUNCTION have been mentioned throughout these release notes. The following list describes all of the new SHADINGENVIRONMNET inputs supported by 4K ACP systems.

User Abort: Input <8> accepts any message and causes an abort to occur for the current hidden-line rendering. Sometimes the calligraphic hidden-line algorithm can take a long time to run to completion. This input allows you to exit before the rendering is complete.

Refresh/Overlay Control: Input <9> accepts a Boolean which determines whether the screen is to be cleared with the current background color before the rendering is done, or whether the requested rendering will overlay the object already displayed on the raster screen.

Sending a TRUE to input <9> of SHADINGENVIRONMENT causes the current rendered object to be displayed on top of the current picture displayed on the the raster screen. Sending a FALSE to input <9> of SHADINGENVIRONMENT causes the screen to be cleared with the current background color before the rendering is done.

Color by Vertex Control: Input <10> accepts a Boolean which turns off (or on) the vertex colors. The use of the color at the vertex rather than the color specified in the ATTRIBUTES command is controlled by sending to this input. To use the vertex colors defined this way rather than the color defined in the ATTRIBUTES, send TRUE to input <10>. Send FALSE to this input to return to using the color specified by the ATTRIBUTES command. (This feature is for 4K ACP systems only). The default is false.

Opaque (Transparency) Control: Input <11> accepts a Boolean which allows you to turn off (or on) the transparency assigned to the polygon with the OPAQUE clause of the attribute command. To render polygons as transparent, at some time prior to the rendering, you must send a TRUE to input <11>. Sending a FALSE to input <11> will cause all objects to be rendered as fully opaque, regardless of their attributes.

Specular Highlight Control: Input <12> accepts a Boolean which allows you to turn on (or off) specular highlights. Flat, Gouraud, and Phong shading all use the same shading equation, so for all types of shading, multiple light sources are processed in each case and specular highlights are calculated. Specular highlights may appear strange in Gouraud or flat shading. In Gouraud shading, the highlights may cause bright horizontal bands to appear inside the polygons.

Blending (near in Z surfaces) Control: Input <13> accepts a Boolean which turns off (or on) the color-blending used for correct spherical rendering. Sending a TRUE to input <13> turns ON this special color blending. Sending a FALSE to input <13> turns OFF special color-blending. (This feature is for 4K ACP systems only.) The default is false.

Spheres and Lines Attribute Table Update: Input <14> accepts the name of a vector list to update attributes for spheres and lines (4K ACP only). The spheres and lines PS 340 enhancement is explained in Section 4.3.

Raster Lines Z-value Control: Input <15> accepts a real number in the range of 0-1 which is added to the z-values of lines in raster renderings. Sending a 0 to this input will leave lines in their original z position. Sending a 1 to this input will force lines to be in front of everything else in the image. This feature may be desirable when rendering lines exactly along polygon edges. Leaving lines at their original z-values will cause obscurity problems with the edges of the the polygons. By adding an offset to the lines' z values, this obscurity problem is more easily resolved. (This features is for 4K ACP only.) The default is 0.

## 4.3 Spheres and Lines Capabilities

Spherical rendering (primarily used in molecular modeling) and raster-lines (primarily used for labeling) have now been incorporated with the the standard PS 340 raster rendering capabilities supported under 4K ACP systems. Spheres and raster lines are represented as vector lists instead of an explicit PS 300 data type. Spheres are shaded consistent with the Phong shading style, allowing multiple colored light sources, specular reflections, and depth cueing.

If "wash" shading is selected for polygons, spheres in the same rendering will be rendered with just one light source at the default position (straight on).

Since spheres are represented as vector lists, no color interpolation or transparency is supported. Lines are rendered with their defined color with no shading, but may have depth-cueing applied. Hidden-element removal with spheres, lines, and polygons has been accomplished with a common z-buffer algorithm.

- Defining Raster Spheres and Lines

  Since there are no explicit PS 300 data types for representing spheres or raster lines, you do not place sphere or raster-line data under a rendering operation node.

  To display line data on the raster display, a tabulated vector list (3D tabulated vector) must be created. The "P" and "L" indicators specify the "moves" and "draws" for raster-line renderings just as they do for calligraphic display. For spherical data, a dots or itemized vector list (3D tabulated vector) must be created where each x, y, z is interpreted as a spherical center.

These vector lists must be tied to F:XFORMDATA functions which are connected directly to inputs in the SOLID_RENDERING node. Inputs <3>, <4>, and <5> have been added to the rendering node to accommodate sphere and line data. Input <3> of the rendering node accepts a transformed vector list (from output<1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering. Similarly, input <4> of the rendering node accepts a transformed vector list and interprets each vector as an x, y, z spherical center for raster rendering. Input <5> of the rendering node accepts the original vector list to enable accurate scaling of the rendering. These inputs are explained in detail in the following sections.

- Attribute Table

The attributes for lines (color) and spheres (radius, color, diffuse, specular) are stored in a default table created at system boot up. This table can be modified via <input> 14 of the SHADINGENVIRONMENT function. The table has the following components:

Hue Saturation Intensity Radius Diffuse Specular

Hue is a real number in the range 0 to 360. Saturation and intensity are real numbers in the range 0 to 1. Radius is a real number greater than 0. Diffuse is a real number in the range 0 to 1. Specular is an integer in the range 0 to 255.

The table is initialized as follows:

| INDEX | Hue | Sat | Intensity | Radius | Diffuse | Specular | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0.5 | 1.8 | 0.7 | 4 | (Grey) |
| 1 | 0 | 0 | 1 | 1.2 | 0.7 | 4 | (White) |
| 2 | 120 | 1 | 1 | 1.35 | 0.7 | 4 | (Red) |
| 3 | 240 | 1 | 1 | 1.8 | 0.7 | 4 | (Green) |
| 4 | 0 | 1 | 1 | 1.8 | 0.7 | 4 | (Blue) |
| 5 | 180 | 1 | 1 | 1.7 | 0.7 | 4 | (Yellow) |
| 6 | 0 | 0 | 0.7 | 1.8 | 0.7 | 4 | (Grey) |
| 7 | 300 | 1 | 1 | 2.15 | 0.7 | 4 | (Cyan) |
| 8 | 60 | 1 | 1 | 1.8 | 0.7 | 4 | (Magenta) |
| 9–127 | 0 | 0 | 1 | 1.8 | 0.7 | 4 | (White) |

Spheres use all six of these components. Lines use only the hue, saturation, and intensity components.

The t field of each 3D tabulated vector is used as an index into this table. The table contains 128 entries (0–127).

For example, the following vector list represents three spheres with the color indicated.

```
SPHERE := VECtor_list  TABulated N = 3
   P  1,2,3    t = 5      {yellow sphere}
   L  4,5,6    t = 6      {grey sphere}
   L  7,8,9    t = 7      {cyan sphere}
;
```

The following example represents a square with sides of the indicated colors.

```
RASTERLINE := VEC TAB N = 5
   P  0,1,0
   L  0,0,0    t = 5      {yellow}
   L  1,0,0    t = 2      {red}
   L  1,1,0    t = 3      {green}
   L  0,1,0    t = 4      {blue}
;
```

**NOTE**

Lines use the tabulated index of the point drawn "to" and not the point drawn "from." Thus, the tabulated index of position vectors is ignored and may be omitted.

- Updating the Attribute Table

The attribute table may be updated by encoding the table entries into a PS 300 tabulated vector list and then sending the name of the vector list to <14>SHADINGENVIRONMENT. The six table components are encoded into two consecutive 3D vectors of the vector list. Hue, saturation, and intensity are encoded into the first x, y, z, respectively. Radius, diffuse, and specular are encoded into the second x, y, z, respectively. The table index is encoded into the t field of the second vector.

For example the following vector list would be used to update attribute table entry 5:

```
ATTRIBUTE_TABLE := VEC TAB N = 2
    150,0.5,1     5.0,0.3,2      t = 5;
```

Updating would be accomplished by the command:

```
@@ SEND 'ATTRIBUTE_TABLE' TO <14>SHADINGENVIRONMENT;
```

Note that more than one table entry may be encoded into a vector list. The following vector list would be used to update attribute table entries 5, 6, and 7:

```
ATTRIBUTE_TABLE := VEC TAB N = 6
   0,1,1       2.0,0.5,4      t= 5
   120,1,1     4.0,0.8,9      t = 6
   240,1,1     3.0,0.3,2      t = 7
;
```

If you change the attribute table (using the above command) to the image of the three atoms defined in the example on the previous page, the yellow, grey, and cyan spheres would change to a small blue sphere, a large, dull, red sphere, and a shiny, medium-sized green sphere.

- Spherical Radii Scaling

To apply the correct radii scaling factor, the rendering node must examine the original or untransformed data. You must supply the name of the vector list representing the spherical data on input <5> of the rendering node. The name is sent as a string in single quotes.

- Sphere and Line Constraints

Window Restriction - For spheres to be rendered correctly, a cubical orthographic projection must be used (i.e., WINDOW command). Spherical renderings with perspective projections or any other non-cubical orthographic projections will not be displayed correctly.

Viewport Restriction - If spheres are to be rendered in conjunction with either lines or polygons, then only the following raster viewport should be used:

```
     Xleft =  64            Xright = 575
   Ybottom = -32              Ytop = 479
```

The command "SEND V3D(64,-32,511) to <3>SHADINGENVIRONMENT;" will set this raster viewport.

Lines and/or polygons can be rendered correctly in any raster viewport. Spheres alone can be rendered correctly in any raster viewport.

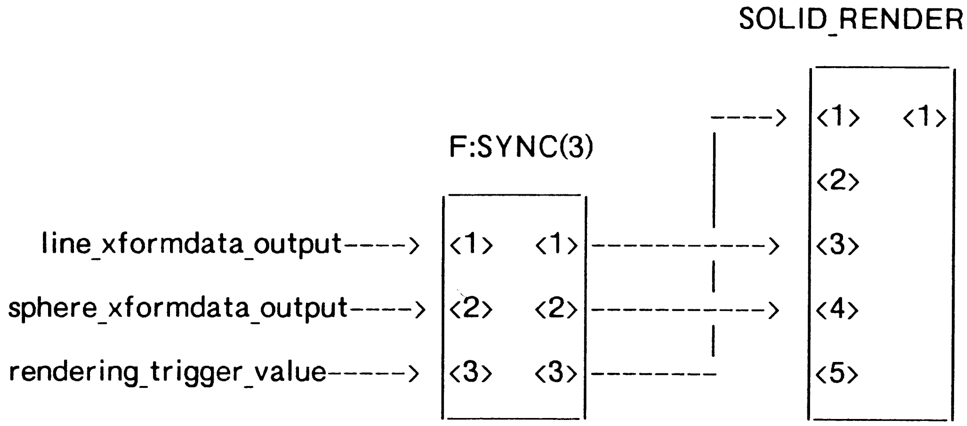
- Edge-smoothing Mode for Spheres and Lines

  Input <5> of SHADINGENVIRONMENT accepts an integer defining the level of edge-smoothing. For lines and spheres to be rendered correctly, a 1 or a 2 should be sent to this input. A value of 0 for edge-smoothing may result in incorrect renderings.

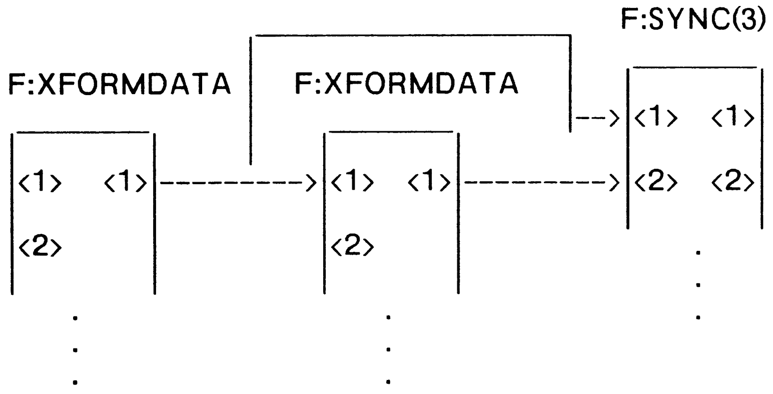
- Function Network Considerations

  Two potential timing problems exist with triggering the rendering node. Input <1> of the rendering node is the only active input. Inputs <3> and <4> which accept transformed data for rendering lines and spheres are constant inputs. (These inputs are constant inputs and initialized to NIL so that application programs written prior to A2.V01 will work without any modifications.) Since inputs <3> and <4> are constant inputs, you must guarantee that they have updated before the trigger is sent to input <1> of the rendering node. This is accomplished using the F:SYNC(n) function.

  F:SYNC(n) sends its outputs in the order 1 to n. By connecting outputs 1 to n-1 of F:SYNC(n) to the constant inputs of the rendering node and by connecting output n of ⊢:SYNC(n) to input <1> of the rendering node, you can guarantee that the constant inputs will be updated before the rendering node is triggered. This is shown in the following example.



  The second potential timing problem deals with the triggering of F:XFORMDATA. An instance of F:XFORMDATA must not be triggered while it or any other instance of F:XFORMDATA is still active. Thus when using multiple instances of F:XFORMDATA, one instance should be used as the trigger for the next. (See the following example.)



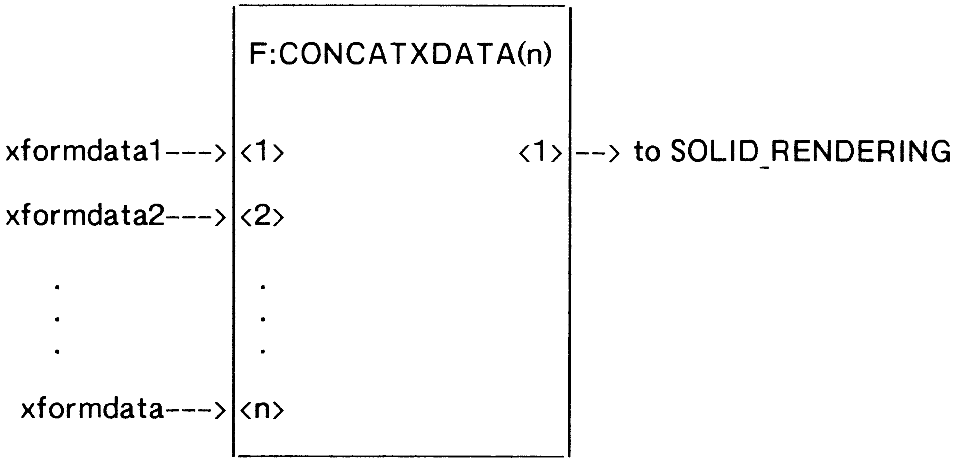
- Naming Transformed Data

You must be aware of one other restriction when using F:XFORMDATA. Input <3> of F:XFORMDATA typically allows you to specify a name for the transformed data. However when using F:XFORMDATA in conjunction with a rendering node, this input must be left blank.

**CAUTION**

Naming the transformed data and then sending it to a rendering node, will result in a system failure.

F:CONCATXDATA(n) accepts up to 127 transformed vector lists (output from XFORMDATA functions) and concatenates them into a single transformed vector list. It is used to avoid the maximum vector restriction imposed on the output of F:XFORMDATA. The XFORMDATA function will return a maximum of 2048 vectors. This restriction passes on to the rendering node since the output of the XFORMDATA function is normally connected directly to the rendering node. To obtain a rendering of greater than 2048 vectors (or spheres), the output of multiple instances of XFORMDATA must be concatenated into a single transformed vector list which can then be sent to the rendering node.

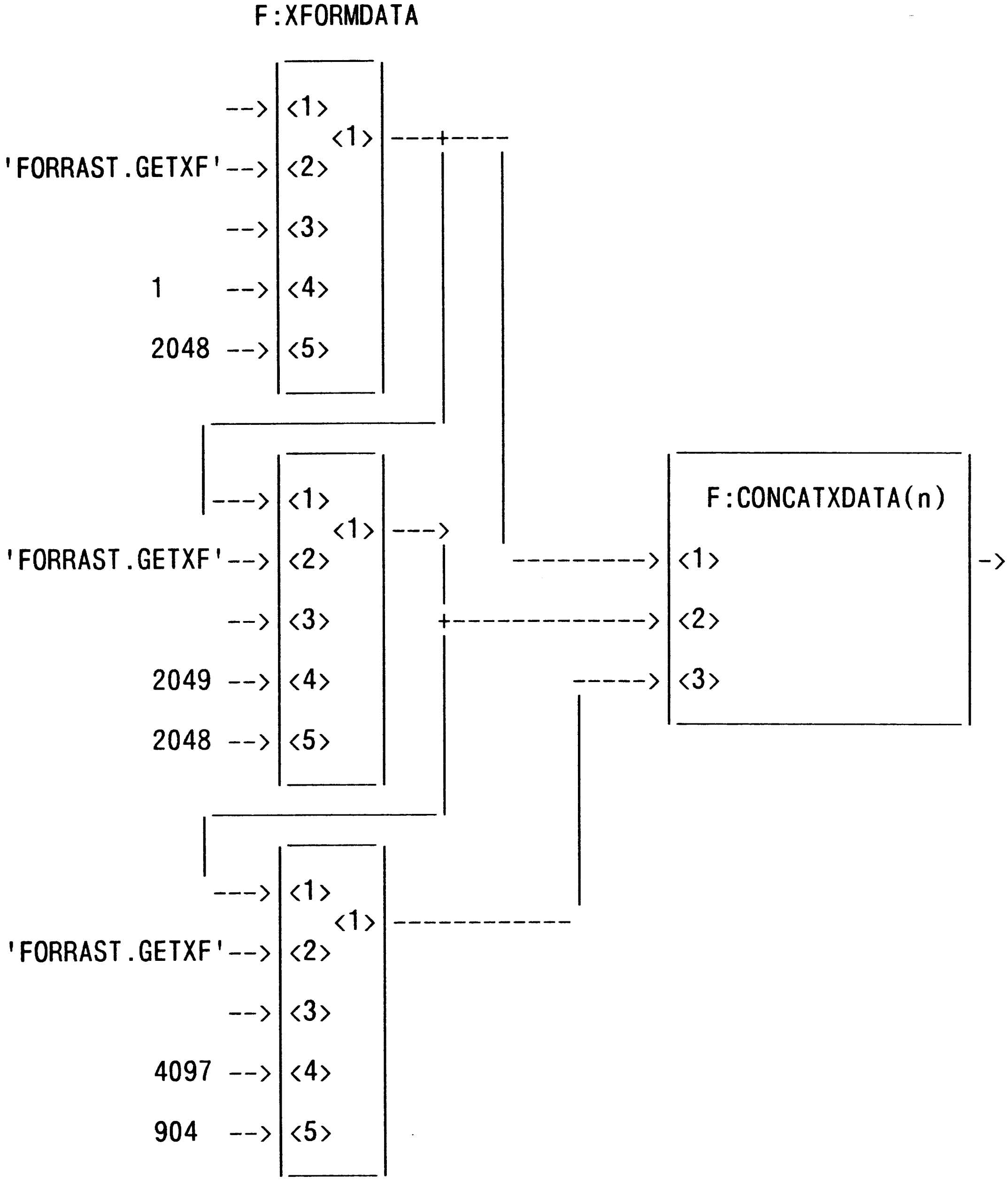


As previously discussed, multiple instances of F:XFORMDATA must be linked together for triggering purposes to ensure that one instance completes before the next one commences.

For example, assume that in the following network, the vector list SPHERES contains 5,000 vectors.

```
FORRAST := BEGIN_STRUCTURE
    GETXF := XFORM VEC;
    INSTANCE OF SPHERES;
END_STRUCTURE;
```

One instance of XFORMDATA could retrieve the first 2048 transformed vectors of MOLECULE (vectors 1-2048). A second execution of XFORMDATA could retrieve the second 2048 transformed vectors (vectors 2049 - 4096). And a third execution of XFORMDATA could retrieve the last 904 vectors (vectors 4097 - 5000). An illustration of this network follows.


F:XFORMDATA
<1>
<1>
'FORRAST.GETXF'
<2>
<3>
1
<4>
2048
<5>
<1>
<1>
'FORRAST.GETXF'
<2>
<3>
2049
<4>
2048
<5>
F:CONCATXDATA(n)
<1>
<2>
<3>
<1>
<1>
'FORRAST.GETXF'
<2>
<3>
4097
<4>
904
<5>

## 5. PROGRAMMING EXAMPLES

Following are two programming examples showing color by vertex and transparency specifications and spheres and lines rendering on the raster screen. These examples are contained on the magnetic distribution tape included with this release.

### 5.1 Specifying Vertex Color

The first polygon, the top of the cylinder, has no normals or vertex colors specified. Since no vertex colors have been specified, and no attribute is given either, the color of the polygon will be given the default attribute which is white.

The second polygon, the bottom of the cylinder, has no normals either. Only vertex colors have been specified. Since no normals are specified the normals default to the normal of the polygon (computed in the PS 340) and no smoothing is done across either of these first two polygons. Since an attribute has been given for this polygon, this polygon will be rendered with a flat red color if the vertex colors have been globally turned off (by sending a FALSE to input <10>SHADINGENVIRONMENT). If vertex colors have been turned on, this polygon will exhibit all the colors present in the color wheel, all blended smoothly with each other and interpolating to a shade of light-grey in the middle of the cylinder.

The third polygon, one of the sides of the cylinder, has soft edges (S) for producing hidden-line pictures on the calligraphic display. Normals are specified so this polygon will be rendered smoothly if you render this with the Gouraud or Phong shading style. The vertex colors have also been specified. Notice that only the hue value has been given, the saturation and intensity will default to 1.0. If vertex colors have been turned off, the polygon will be rendered with the attribute that it has been given, cyan.

The fourth polygon, an adjacent side, also has normals and vertex colors specified. In this case, only the hue and saturation values have been given. The intensity value will default to 1.0. If vertex colors have been turned off, the polygon will be rendered with the attribute that it has been given, magenta. Notice that the attribute magenta is transparent with an opaque specification of 0.5. This means that if transparency is turned ON (by sending TRUE to input <11> of SHADINGENVIRONMENT), this polygon will appear transparent with a filtering color specified by either the attribute magenta or the vertex colors (depending on whether or not vertex colors have been turned off).

The rest of the polygons all have normals, and vertex colors fully specified with hue, saturation, and intensity values.

```
BLUE       := ATTRIBUTES COLOR 0;
MAGENTA    := ATTRIBUTES COLOR 60   DIFFUSE 0.8 SPECULAR 4 OPAQUE 0.5;
PURPLE     := ATTRIBUTES COLOR 90;
RED        := ATTRIBUTES COLOR 120;
ORANGE     := ATTRIBUTES COLOR 150  DIFFUSE 0.8 SPECULAR 4 OPAQUE 0.5;
YELLOW     := ATTRIBUTES COLOR 180;
GREEN      := ATTRIBUTES COLOR 240  DIFFUSE 0.8 SPECULAR 4 OPAQUE 0.5;
TURQUOISE  := ATTRIBUTES COLOR 270;
CYAN       := ATTRIBUTES COLOR 300;
WHITE      := ATTRIBUTES COLOR 0, 0, 1;
GREY       := ATTRIBUTES COLOR 0, 0, 0.5;
BLACK      := ATTRIBUTES COLOR 0, 0, 0;


CYLINDER :=
    POLYGON
          1.00000,   0.00000,   0.50000
          0.95106,   0.30902,   0.50000
          0.80902,   0.58779,   0.50000
          0.58779,   0.80902,   0.50000
          0.30902,   0.95106,   0.50000
          0.00000,   1.00000,   0.50000
         -0.30902,   0.95106,   0.50000
         -0.58779,   0.80902,   0.50000
         -0.80902,   0.58779,   0.50000
         -0.95106,   0.30902,   0.50000
         -1.00000,   0.00000,   0.50000
         -0.95106,  -0.30902,   0.50000
         -0.80902,  -0.58779,   0.50000
         -0.58779,  -0.80902,   0.50000
         -0.30902,  -0.95106,   0.50000
          0.00000,  -1.00000,   0.50000
          0.30902,  -0.95106,   0.50000
          0.58779,  -0.80902,   0.50000
          0.80902,  -0.58779,   0.50000
          0.95106,  -0.30902,   0.50000
```

```
WITH ATTRIBUTES RED
  POLYGON
        0.95106,  -0.30902,  -0.50000 C 288.00003,   1.00000,   1.00000
        0.80902,  -0.58779,  -0.50000 C 306.00006,   1.00000,   1.00000
        0.58779,  -0.80902,  -0.50000 C 324.00003,   1.00000,   1.00000
        0.30902,  -0.95106,  -0.50000 C 342.00003,   1.00000,   1.00000
        0.00000,  -1.00000,  -0.50000 C 360.00003,   1.00000,   1.00000
       -0.30902,  -0.95106,  -0.50000 C  17.99998,   1.00000,   1.00000
       -0.58779,  -0.80902,  -0.50000 C  36.00000,   1.00000,   1.00000
       -0.80902,  -0.58779,  -0.50000 C  53.99999,   1.00000,   1.00000
       -0.95106,  -0.30902,  -0.50000 C  71.99998,   1.00000,   1.00000
       -1.00000,   0.00000,  -0.50000 C  90.00001,   1.00000,   1.00000
       -0.95106,   0.30902,  -0.50000 C 108.00002,   1.00000,   1.00000
       -0.80902,   0.58779,  -0.50000 C 126.00001,   1.00000,   1.00000
       -0.58779,   0.80902,  -0.50000 C 144.00000,   1.00000,   1.00000
       -0.30902,   0.95106,  -0.50000 C 162.00000,   1.00000,   1.00000
        0.00000,   1.00000,  -0.50000 C 180.00000,   1.00000,   1.00000
        0.30902,   0.95106,  -0.50000 C 198.00002,   1.00000,   1.00000
        0.58779,   0.80902,  -0.50000 C 216.00000,   1.00000,   1.00000
        0.80902,   0.58779,  -0.50000 C 234.00002,   1.00000,   1.00000
        0.95106,   0.30902,  -0.50000 C 252.00002,   1.00000,   1.00000
        1.00000,   0.00000,  -0.50000 C 270.00000,   1.00000,   1.00000

WITH ATTRIBUTES CYAN
  POLYGON
    S   1.00000,   0.00000,  -0.50000 N   1.97538,   0.00000,  -1.00000
                                      C 270.00000
        0.95106,   0.30902,  -0.50000 N   1.87869,   0.61042,  -1.00000
                                      C 252.00002
    S   0.95106,   0.30902,   0.50000 N   1.87869,   0.61042,   1.00000
                                      C 252.00002
        1.00000,   0.00000,   0.50000 N   1.97538,   0.00000,   1.00000
                                      C 270.00000

WITH ATTRIBUTES MAGENTA
  POLYGON
        0.95106,   0.30902,  -0.50000 N   1.87869,   0.61042,  -1.00000
                                      C 252.00002,   1.00000
        0.80902,   0.58779,  -0.50000 N   1.59811,   1.16110,  -1.00000
                                      C 234.00002,   1.00000
    S   0.80902,   0.58779,   0.50000 N   1.59811,   1.16110,   1.00000
                                      C 234.00002,   1.00000
        0.95106,   0.30902,   0.50000 N   1.87869,   0.61042,   1.00000
                                      C 252.00002,   1.00000
```

```
WITH ATTRIBUTES ORANGE
  POLYGON
           0.80902,    0.58779,   -0.50000 N   1.59811,    1.16110,   -1.00000
                                           C 234.00002,    1.00000,    1.00000
           0.58779,    0.80902,   -0.50000 N   1.16110,    1.59811,   -1.00000
                                           C 216.00000,    1.00000,    1.00000
       S   0.58779,    0.80902,    0.50000 N   1.16110,    1.59811,    1.00000
                                           C 216.00000,    1.00000,    1.00000
           0.80902,    0.58779,    0.50000 N   1.59811,    1.16110,    1.00000
                                           C 234.00002,    1.00000,    1.00000

WITH ATTRIBUTES GREEN
  POLYGON
           0.58779,    0.80902,   -0.50000 N   1.16110,    1.59811,   -1.00000
                                           C 216.00000,    1.00000,    1.00000
           0.30902,    0.95106,   -0.50000 N   0.61042,    1.87869,   -1.00000
                                           C 198.00002,    1.00000,    1.00000
       S   0.30902,    0.95106,    0.50000 N   0.61042,    1.87869,    1.00000
                                           C 198.00002,    1.00000,    1.00000
           0.58779,    0.80902,    0.50000 N   1.16110,    1.59811,    1.00000
                                           C 216.00000,    1.00000,    1.00000

WITH ATTRIBUTES YELLOW
  POLYGON
           0.30902,    0.95106,   -0.50000 N   0.61042,    1.87869,   -1.00000
                                           C 198.00002,    1.00000,    1.00000
           0.00000,    1.00000,   -0.50000 N   0.00000,    1.97538,   -1.00000
                                           C 180.00000,    1.00000,    1.00000
       S   0.00000,    1.00000,    0.50000 N   0.00000,    1.97538,    1.00000
                                           C 180.00000,    1.00000,    1.00000
           0.30902,    0.95106,    0.50000 N   0.61042,    1.87869,    1.00000
                                           C 198.00002,    1.00000,    1.00000

WITH ATTRIBUTES CYAN
  POLYGON
           0.00000,    1.00000,   -0.50000 N   0.00000,    1.97538,   -1.00000
                                           C 180.00000,    1.00000,    1.00000
          -0.30902,    0.95106,   -0.50000 N  -0.61042,    1.87869,   -1.00000
                                           C 162.00000,    1.00000,    1.00000
       S  -0.30902,    0.95106,    0.50000 N  -0.61042,    1.87869,    1.00000
                                           C 162.00000,    1.00000,    1.00000
           0.00000,    1.00000,    0.50000 N   0.00000,    1.97538,    1.00000
                                           C 180.00000,    1.00000,    1.00000
```

```
WITH ATTRIBUTES MAGENTA
  POLYGON
       -0.30902,    0.95106,   -0.50000 N  -0.61042,    1.87869,   -1.00000
                                        C 162.00000,    1.00000,    1.00000
       -0.58779,    0.80902,   -0.50000 N  -1.16110,    1.59811,   -1.00000
                                        C 144.00000,    1.00000,    1.00000
     S -0.58779,    0.80902,    0.50000 N  -1.16110,    1.59811,    1.00000
                                        C 144.00000,    1.00000,    1.00000
       -0.30902,    0.95106,    0.50000 N  -0.61042,    1.87869,    1.00000
                                        C 162.00000,    1.00000,    1.00000


WITH ATTRIBUTES ORANGE
  POLYGON
       -0.58779,    0.80902,   -0.50000 N  -1.16110,    1.59811,   -1.00000
                                        C 144.00000,    1.00000,    1.00000
       -0.80902,    0.58779,   -0.50000 N  -1.59811,    1.16110,   -1.00000
                                        C 126.00001,    1.00000,    1.00000
     S -0.80902,    0.58779,    0.50000 N  -1.59811,    1.16110,    1.00000
                                        C 126.00001,    1.00000,    1.00000
       -0.58779,    0.80902,    0.50000 N  -1.16110,    1.59811,    1.00000
                                        C 144.00000,    1.00000,    1.00000


WITH ATTRIBUTES GREEN
  POLYGON
       -0.80902,    0.58779,   -0.50000 N  -1.59811,    1.16110,   -1.00000
                                        C 126.00001,    1.00000,    1.00000
       -0.95106,    0.30902,   -0.50000 N  -1.87869,    0.61043,   -1.00000
                                        C 108.00002,    1.00000,    1.00000
     S -0.95106,    0.30902,    0.50000 N  -1.87869,    0.61043,    1.00000
                                        C 108.00002,    1.00000,    1.00000
       -0.80902,    0.58779,    0.50000 N  -1.59811,    1.16110,    1.00000
                                        C 126.00001,    1.00000,    1.00000


WITH ATTRIBUTES YELLOW
  POLYGON
       -0.95106,    0.30902,   -0.50000 N  -1.87869,    0.61043,   -1.00000
                                        C 108.00002,    1.00000,    1.00000
       -1.00000,    0.00000,   -0.50000 N  -1.97538,    0.00000,   -1.00000
                                        C  90.00001,    1.00000,    1.00000
     S -1.00000,    0.00000,    0.50000 N  -1.97538,    0.00000,    1.00000
                                        C  90.00001,    1.00000,    1.00000
       -0.95106,    0.30902,    0.50000 N  -1.87869,    0.61043,    1.00000
                                        C 108.00002,    1.00000,    1.00000
```

```
WITH ATTRIBUTES CYAN
  POLYGON
       -1.00000,   0.00000,  -0.50000 N -1.97538,   0.00000,  -1.00000
                                      C 90.00001,   1.00000,   1.00000
       -0.95106,  -0.30902,  -0.50000 N -1.87870,  -0.61042,  -1.00000
                                      C 71.99998,   1.00000,   1.00000
     S -0.95106,  -0.30902,   0.50000 N -1.87870,  -0.61042,   1.00000
                                      C 71.99998,   1.00000,   1.00000
       -1.00000,   0.00000,   0.50000 N -1.97538,   0.00000,   1.00000
                                      C 90.00001,   1.00000,   1.00000


WITH ATTRIBUTES MAGENTA
  POLYGON
       -0.95106,  -0.30902,  -0.50000 N -1.87870,  -0.61042,  -1.00000
                                      C 71.99998,   1.00000,   1.00000
       -0.80902,  -0.58779,  -0.50000 N -1.59811,  -1.16110,  -1.00000
                                      C 53.99999,   1.00000,   1.00000
     S -0.80902,  -0.58779,   0.50000 N -1.59811,  -1.16110,   1.00000
                                      C 53.99999,   1.00000,   1.00000
       -0.95106,  -0.30902,   0.50000 N -1.87870,  -0.61042,   1.00000
                                      C 71.99998,   1.00000,   1.00000


WITH ATTRIBUTES ORANGE
  POLYGON
       -0.80902,  -0.58779,  -0.50000 N -1.59811,  -1.16110,  -1.00000
                                      C 53.99999,   1.00000,   1.00000
       -0.58779,  -0.80902,  -0.50000 N -1.16110,  -1.59811,  -1.00000
                                      C 36.00000,   1.00000,   1.00000
     S -0.58779,  -0.80902,   0.50000 N -1.16110,  -1.59811,   1.00000
                                      C 36.00000,   1.00000,   1.00000
       -0.80902,  -0.58779,   0.50000 N -1.59811,  -1.16110,   1.00000
                                      C 53.99999,   1.00000,   1.00000
WITH ATTRIBUTES GREEN
  POLYGON
       -0.58779,  -0.80902,  -0.50000 N -1.16110,  -1.59811,  -1.00000
                                      C 36.00000,   1.00000,   1.00000
       -0.30902,  -0.95106,  -0.50000 N -0.61043,  -1.87869,  -1.00000
                                      C 17.99998,   1.00000,   1.00000
     S -0.30902,  -0.95106,   0.50000 N -0.61043,  -1.87869,   1.00000
                                      C 17.99998,   1.00000,   1.00000
       -0.58779,  -0.80902,   0.50000 N -1.16110,  -1.59811,   1.00000
                                      C 36.00000,   1.00000,   1.00000
```

```
WITH ATTRIBUTES YELLOW
  POLYGON
       -0.30902,  -0.95106,  -0.50000 N  -0.61043,  -1.87869,  -1.00000
                                      C  17.99998,   1.00000,   1.00000
        0.00000,  -1.00000,  -0.50000 N   0.00000,  -1.97538,  -1.00000
                                      C 360.00003,   1.00000,   1.00000
     S  0.00000,  -1.00000,   0.50000 N   0.00000,  -1.97538,   1.00000
                                      C 360.00003,   1.00000,   1.00000
       -0.30902,  -0.95106,   0.50000 N  -0.61043,  -1.87869,   1.00000
                                      C  17.99998,   1.00000,   1.00000


WITH ATTRIBUTES CYAN
  POLYGON
        0.00000,  -1.00000,  -0.50000 N   0.00000,  -1.97538,  -1.00000
                                      C 360.00003,   1.00000,   1.00000
        0.30902,  -0.95106,  -0.50000 N   0.61042,  -1.87870,  -1.00000
                                      C 342.00003,   1.00000,   1.00000
     S  0.30902,  -0.95106,   0.50000 N   0.61042,  -1.87870,   1.00000
                                      C 342.00003,   1.00000,   1.00000
        0.00000,  -1.00000,   0.50000 N   0.00000,  -1.97538,   1.00000
                                      C 360.00003,   1.00000,   1.00000


WITH ATTRIBUTES MAGENTA
  POLYGON
        0.30902,  -0.95106,  -0.50000 N   0.61042,  -1.87870,  -1.00000
                                      C 342.00003,   1.00000,   1.00000
        0.58779,  -0.80902,  -0.50000 N   1.16110,  -1.59811,  -1.00000
                                      C 324.00003,   1.00000,   1.00000
     S  0.58779,  -0.80902,   0.50000 N   1.16110,  -1.59811,   1.00000
                                      C 324.00003,   1.00000,   1.00000
        0.30902,  -0.95106,   0.50000 N   0.61042,  -1.87870,   1.00000
                                      C 342.00003,   1.00000,   1.00000


WITH ATTRIBUTES ORANGE
  POLYGON
        0.58779,  -0.80902,  -0.50000 N   1.16110,  -1.59811,  -1.00000
                                      C 324.00003,   1.00000,   1.00000
        0.80902,  -0.58779,  -0.50000 N   1.59811,  -1.16110,  -1.00000
                                      C 306.00006,   1.00000,   1.00000
     S  0.80902,  -0.58779,   0.50000 N   1.59811,  -1.16110,   1.00000
                                      C 306.00006,   1.00000,   1.00000
        0.58779,  -0.80902,   0.50000 N   1.16110,  -1.59811,   1.00000
                                      C 324.00003,   1.00000,   1.00000
```

```
WITH ATTRIBUTES GREEN
  POLYGON
            0.80902,  -0.58779,  -0.50000 N   1.59811,  -1.16110,  -1.00000
                                          C 306.00006,   1.00000,   1.00000
            0.95106,  -0.30902,  -0.50000 N   1.87869,  -0.61043,  -1.00000
                                          C 288.00003,   1.00000,   1.00000
        S   0.95106,  -0.30902,   0.50000 N   1.87869,  -0.61043,   1.00000
                                          C 288.00003,   1.00000,   1.00000
            0.80902,  -0.58779,   0.50000 N   1.59811,  -1.16110,   1.00000
                                          C 306.00006,   1.00000,   1.00000


WITH ATTRIBUTES YELLOW
  POLYGON
            0.95106,  -0.30902,  -0.50000 N   1.87869,  -0.61043,  -1.00000
                                          C 288.00003,   1.00000,   1.00000
            1.00000,   0.00000,  -0.50000 N   1.97538,   0.00000,  -1.00000
                                          C 270.00003,   1.00000,   1.00000
            1.00000,   0.00000,   0.50000 N   1.97538,   0.00000,   1.00000
                                          C 270.00003,   1.00000,   1.00000
            0.95106,  -0.30902,   0.50000 N   1.87869,  -0.61043,   1.00000
                                          C 288.00003,   1.00000,   1.00000
;
```

## 5.2 Spheres and Lines Programming Example

Following is a programming example of a function network displaying spheres and lines on the raster screen. This network will load and prepare for rendering one green line segment and four spheres (one red, one blue, and two green). The spheres are a portion of a molecular model. Sections of this network which are particularly relevant to lines and spheres are noted with several asterisks (*********).

```
initialize;

{reserve memory for rendering is needed only once per session and is
typically included in the Site.Dat file executed at boot time}

reserve_working_storage 300000;

{define a sectioning plane which can be rotated independently}

spattributes := attributes;

sect :=begin_s
       sectioning_plane;
       trans := translate by 0,0,0;
       rot   := scale by 1;
       with attributes spattributes
       polygon -0.9,-0.9,0.0 -0.9,0.9,0.0 0.9,0.9,0.0 0.9,-0.9,0.0
       polygon 0.1,0.0,0.0 0.1,0.0,-0.3 0.15,0.0,-0.3 0.0,0.0,-0.45
               -0.15,0.0,-0.3 -0.1,0.0,-0.3 -0.1,0.0,0.0
       polygon 0.0,0.1,0.0 0.0,0.1,-0.3 0.0,0.15,-0.3 0.0,0.0,-0.45
               0.0,-0.15,-0.3 0.0,-0.1,-0.3 0.0,-0.1,0.0;
       end_s;

{define a light which can be rotated independently}

sunset := begin_structure
          persp := fov 90 front=2.2 back=3.6;
          look at 0,0,0 from 0,0,-3;
          set depth_clipping off;
          rot := scale by 1;
          vector 0,0,-.9 0,0,0;
          instance sun;
          translate 0,0,-.9;
          rational polynomial .2,0,8 -.2,-.2,-8 0,.1,4 chords=15;
          rational polynomial .2,0,-8 -.2,-.2,8 0,.1,-4 chords=15;
          vector separate n=15 -.1,0 -.05,0 .05,0 .1,0
               0,-.1 0,-.05 0,.05 0,.1
               -.0707,-.0707 -.0354,-.0354 .0354,.0354 .0707,.0707
               -.0707,.0707 -.0354,.0354 .0354,-.0354 .0707,-.0707;
          end_structure;
```

```
sun := illumination 0,0,-1;

{define a light which can be rotated with the object}

moonset :=begin_structure
           set depth_clipping off;
           rot := scale by 1;
           vector 0,0,-.9 0,0,0;
           instance moon;
           translate 0,0,-.9;
           rational polynomial .2,0,4 -.2,-.2,-4 0,.1,2 chords=15;
           rational polynomial .12,0,4 -.12,-.2,-4 0,.1,2 chords=15;
           end_structure;
moon := illumination 0,0,-1;

{set up a place to redisplay a saved hidden-line picture}

disphlview := matrix_4x4 1,0,0,0 0,1,0,0 0,0,0,0 0,0,1,1 then hlview;

{set up initial display structure}

universe := begin_s;
      csm := set condition 1 off;
      set contrast 0;
      if condition 1 on then rainbow;
      if condition 1 off then world;
end_s;

rainbow := begin_s;
      csm2 := set csm on;
      csm3 := set color blending 1;
      inst world;
end_s;

world := begin_s
      bits := set condition 1 on;
      if condition 1 off then disphlview;
      if condition 1 on;
      window x = -1:1 y= -1:1
      front boundary = 19
      back boundary = 21;
      from := LOOK AT 0,0,0 FROM 0,0,-20;

      set depth_clipping on;

      alltrans := translate by 0,0,0;
      allrot   := scale by 1;

{ ********** }
      instance of lines;
      instance of spheres;
{ ********** }

      trans := translate by 0,0,0;
      rot   := scale by 1;
      if condition 2 on then sect;
      rendering := surface;
         { rendering operate node, initially a surface }
      if condition 3 on then sunset;
      if condition 4 on then moonset;
      instance object;
      end_s;
display universe;

{ ********** }
lines := begin_s
    linexform := xform vector;
    trans := translate by 0,0,0;
    rot := scale by 1;
    instance of line_vlist;
end_s;

line_vlist := vec tabulated n = 2
   p -1,-1,0 t = 2
   l  1, 1,0 t = 3
  ;

linexformdata := f:xformdata;
send 'lines.linexform' to <2>linexformdata;    { do not name input<3>! }

spheres := begin_s
    spherexform := xform vector;
    trans := translate by 0,0,0;
    rot := scale by 1;
    instance of sphere_vlist;
end_s;

sphere_vlist := vec tabulated n = 512
   p 0.387,-0.368,-0.051   t=2
   l -0.574,-0.500,-1.155  t=3
   l 0.037,-1.170,1.155    t=3
   l 0.573,1.170,0.398     t=4
 ;

give_up_cpu;
give_up_cpu;
send 'sphere_vlist' to <5>world.rendering;  { original vector list for }
                                            { radii scale factor       }
```

```
spherexformdata := f:xformdata;
send 'spheres.spherexform' to <2>spherexformdata;

{ ********** }
{network to translate object}

a := f:addc;
connect a<1>:<1>world.trans;
connect a<1>:<2>a;
send v3d(0,0,0) to <2>a;

{network to rotate/scale object}

m := f:cmul;
connect m<1>:<1>world.rot;
connect m<1>:<1>m;
send m3d(1,0,0 0,1,0 0,0,1) to <1>m;

{network to translate all}

aall := f:addc;
connect aall<1>:<1>world.alltrans;
connect aall<1>:<2>aall;
send v3d(0,0,0) to <2>aall;

{network to rotate/scale all}

mall := f:cmul;
connect mall<1>:<1>world.allrot;
connect mall<1>:<1>mall;
send m3d(1,0,0 0,1,0 0,0,1) to <1>mall;

{network to translate lines}

alines := f:addc;
connect alines<1>:<1>lines.trans;
connect alines<1>:<2>alines;
send v3d(0,0,0) to <2>alines;

{network to rotate/scale lines}

mlines := f:cmul;
connect mlines<1>:<1>lines.rot;
connect mlines<1>:<1>mlines;
send m3d(1,0,0 0,1,0 0,0,1) to <1>mlines;

{network to translate spheres}

aspheres := f:addc;
connect aspheres<1>:<1>spheres.trans;
connect aspheres<1>:<2>aspheres;
send v3d(0,0,0) to <2>aspheres;
```

```
{network to rotate/scale spheres}

mspheres := f:cmul;
connect mspheres<1>:<1>spheres.rot;
connect mspheres<1>:<1>mspheres;
send m3d(1,0,0 0,1,0 0,0,1) to <1>mspheres;

{network to translate sectioning plane}

a2 := f:addc;
connect a2<1>:<1>sect.trans;
connect a2<1>:<2>a2;
send v3d(0,0,0) to <2>a2;

{network to rotate/scale sectioning plane}

m2 := f:cmul;
connect m2<1>:<1>sect.rot;
connect m2<1>:<1>m2;
send m3d(1,0,0 0,1,0 0,0,1) to <1>m2;

{network to rotate sun}

msun := f:cmul;
connect msun<1>:<1>sunset.rot;
connect msun<1>:<1>msun;
send m3d(1,0,0 0,1,0 0,0,1) to <1>msun;

{network to rotate moon}

mmoon := f:cmul;
connect mmoon<1>:<1>moonset.rot;
connect mmoon<1>:<1>mmoon;
send m3d(1,0,0 0,1,0 0,0,1) to <1>mmoon;

{network selecting original or rendered view}

original := f:constant;
conn original<1>:<1>world.rendering;
send FALSE to <2>original; { to switch to original view }
connect dials<1>:<1>original;
connect dials<2>:<1>original;
connect dials<3>:<1>original;
connect dials<4>:<1>original;
connect dials<5>:<1>original;
connect dials<6>:<1>original;
connect dials<7>:<1>original;
connect dials<8>:<1>original;
```

```
{color network}

tripcolor := f:sync(5);
setup cness true <2>tripcolor;
setup cness true <3>tripcolor;
setup cness true <4>tripcolor;
setup cness true <5>tripcolor;
connect tripcolor<2>:<3>shadingenvironment;
connect tripcolor<3>:<7>shadingenvironment;
connect tripcolor<4>:<3>shadingenvironment;
connect tripcolor<5>:<2>shadingenvironment;
send v3d(0,440,39) to <2>tripcolor;
send false to <3>tripcolor;
send v3d(0,0,0) to <5>tripcolor;

suncolor := f:accumulate;
connect suncolor<1>:<2>sun;
connect suncolor<1>:<2>shadingenvironment;
connect suncolor<1>:<1>tripcolor;
send v3d(0,0,1) to <2>suncolor;
send 0 to <3>suncolor;
send v3d(20,.25,.25) to <4>suncolor;
send v3d(360,1,1) to <5>suncolor;
send v3d(0,0,0) to <6>suncolor;

mooncolor := f:accumulate;
connect mooncolor<1>:<2>moon;
connect mooncolor<1>:<2>shadingenvironment;
connect mooncolor<1>:<1>tripcolor;
send v3d(0,0,1) to <2>mooncolor;
send 0 to <3>mooncolor;
send v3d(20,.25,.25) to <4>mooncolor;
send v3d(360,1,1) to <5>mooncolor;
send v3d(0,0,0) to <6>mooncolor;

backgroundcolor := f:accumulate;
connect backgroundcolor<1>:<2>shadingenvironment;
connect backgroundcolor<1>:<1>tripcolor;
connect backgroundcolor<1>:<5>tripcolor;
send v3d(0,0,0) to <2>backgroundcolor;
send 0 to <3>backgroundcolor;
send v3d(20,.25,.25) to <4>backgroundcolor;
send v3d(360,1,1) to <5>backgroundcolor;
send v3d(0,0,0) to <6>backgroundcolor;

{mux the dials}
```

```
dialmux := f:croute(8);
connect dialmux<1>:<1>a;
connect dialmux<6>:<1>alines;
connect dialmux<7>:<1>aspheres;
connect dialmux<8>:<1>aall;
connect dialmux<2>:<1>a2;
connect dialmux<3>:<1>suncolor;
connect dialmux<4>:<1>mooncolor;
connect dialmux<5>:<1>backgroundcolor;

dialmux2 := f:croute(8);
connect dialmux2<1>:<2>m;
connect dialmux2<6>:<2>mlines;
connect dialmux2<7>:<2>mspheres;
connect dialmux2<8>:<2>mall;
connect dialmux2<2>:<2>m2;
connect dialmux2<3>:<2>msun;
connect dialmux2<4>:<2>mmoon;

{network to translate in x}

tx := f:xvec;
connect tx<1>:<2>dialmux;
connect dials<1>:<1>tx;

{network to translate in y}

ty := f:yvec;
connect ty<1>:<2>dialmux;
connect dials<2>:<1>ty;

{network to translate in z}

tz := f:zvec;
connect tz<1>:<2>dialmux;
connect dials<3>:<1>tz;

{network to scale}

s := f:scale;
connect s<1>:<2>dialmux2;
sa := f:addc;
connect sa<1>:<1>s;
send 1 to <2>sa;
connect dials<4>:<1>sa;

{network to rotate in x}
```

```
rx := f:xrotate;
connect rx<1>:<2>dialmux2;
sx := f:mulc;
connect sx<1>:<1>rx;
send 100 to <2>sx;
connect dials<5>:<1>sx;

{network to rotate in y}

ry := f:yrotate;
connect ry<1>:<2>dialmux2;
sy := f:mulc;
connect sy<1>:<1>ry;
send 100 to <2>sy;
connect dials<6>:<1>sy;

{network to rotate in z}

rz := f:zrotate;
connect rz<1>:<2>dialmux2;
sz := f:mulc;
connect sz<1>:<1>rz;
send -100 to <2>sz;
connect dials<7>:<1>sz;

{network to adjust back clipping plane}

backclip := f:fov;
{ connect backclip<1>:<1>world.persp; }
connect backclip<1>:<1>sunset.persp;
setup cness false <4>backclip;
setup cness true <1>backclip;
send true to <1>backclip;
send 45 to <2>backclip;
send 2.2 to <3>backclip;
backclipaccum := f:accum;
connect backclipaccum<1>:<4>backclip;
connect dials<8>:<1>backclipaccum;
send 3.6 to <2>backclipaccum;
send 0 to <3>backclipaccum;
send 1 to <4>backclipaccum;
send 30 to <5>backclipaccum;
send 2.2 to <6>backclipaccum;

{network to reset transformations}
```

```
rs := f:sync(2);
connect rs<1>:<1>world.trans;
connect rs<1>:<2>a;
connect rs<2>:<1>world.rot;
connect rs<2>:<1>m;
connect rs<2>:<2>rs;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rs;

rsall := f:sync(2);
connect rsall<1>:<1>world.alltrans;
connect rsall<1>:<2>aall;
connect rsall<2>:<1>world.allrot;
connect rsall<2>:<1>mall;
connect rsall<2>:<2>rsall;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rsall;

rslines := f:sync(2);
connect rslines<1>:<1>lines.trans;
connect rslines<1>:<2>alines;
connect rslines<2>:<1>lines.rot;
connect rslines<2>:<1>mlines;
connect rslines<2>:<2>rslines;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rslines;

rsspheres := f:sync(2);
connect rsspheres<1>:<1>spheres.trans;
connect rsspheres<1>:<2>aspheres;
connect rsspheres<2>:<1>spheres.rot;
connect rsspheres<2>:<1>mspheres;
connect rsspheres<2>:<2>rsspheres;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rsspheres;

rs2 := f:sync(2);
connect rs2<1>:<1>sect.trans;
connect rs2<1>:<2>a2;
connect rs2<2>:<1>sect.rot;
connect rs2<2>:<1>m2;
connect rs2<2>:<2>rs2;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rs2;

rssun := f:constant;
connect rssun<1>:<1>msun;
connect rssun<1>:<1>sunset.rot;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rssun;

rsmoon := f:constant;
connect rsmoon<1>:<1>mmoon;
connect rsmoon<1>:<1>moonset.rot;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rsmoon;
```

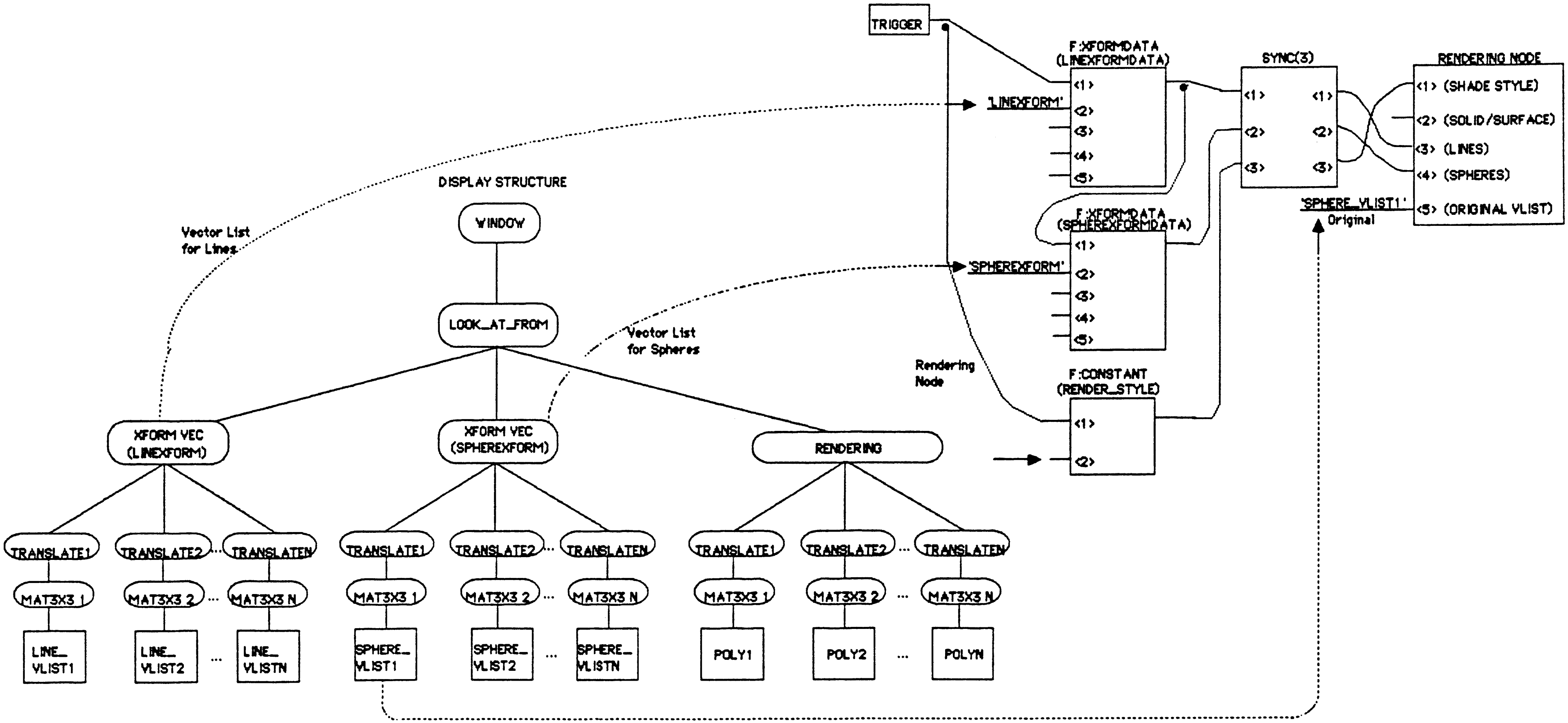
```
r := f:croute(8);
connect r<1>:<1>rs;
connect r<6>:<1>rslines;
connect r<7>:<1>rsspheres;
connect r<8>:<1>rsall;
connect r<2>:<1>rs2;
connect r<3>:<1>rssun;
connect r<4>:<1>rsmoon;

{network to turn bits on and off}

bits := f:constant;
connect bits<1>:<5>world.bits;

{network to send to object or sectioning plane}

waylabel := f:inputs_choose(9);
connect waylabel<1>:<1>flabel12;
send 'OBJECT'   to <1>waylabel;
send 'VECTORS'  to <6>waylabel;
send 'SPHERES'  to <7>waylabel;
send 'ALLPRIMS'  to <8>waylabel;
send 'PLANE' to <2>waylabel;
send 'SUN' to <3>waylabel;
send 'MOON' to <4>waylabel;
send 'BACK' to <5>waylabel;

diallabel := f:sync(9);
connect diallabel<1>:<1>dlabel1;
connect diallabel<1>:<1>diallabel;
connect diallabel<2>:<1>dlabel2;
connect diallabel<2>:<2>diallabel;
connect diallabel<3>:<1>dlabel3;
connect diallabel<3>:<3>diallabel;
connect diallabel<4>:<1>dlabel4;
connect diallabel<4>:<4>diallabel;
connect diallabel<5>:<1>dlabel5;
connect diallabel<5>:<5>diallabel;
connect diallabel<6>:<1>dlabel6;
connect diallabel<6>:<6>diallabel;
connect diallabel<7>:<1>dlabel7;
connect diallabel<7>:<7>diallabel;
connect diallabel<8>:<1>dlabel8;
connect diallabel<8>:<8>diallabel;
```

```
send 'X-TRANS' to <1>diallabel;
send 'X-TRANS' to <1>diallabel;
send 'HUE' to <1>diallabel;
send 'HUE' to <1>diallabel;
send 'HUE' to <1>diallabel;
send 'X-TRANS' to <1>diallabel;
send 'X-TRANS' to <1>diallabel;
send 'X-TRANS' to <1>diallabel;

send 'Y-TRANS' to <2>diallabel;
send 'Y-TRANS' to <2>diallabel;
send 'SAT' to <2>diallabel;
send 'SAT' to <2>diallabel;
send 'SAT' to <2>diallabel;
send 'Y-TRANS' to <2>diallabel;
send 'Y-TRANS' to <2>diallabel;
send 'Y-TRANS' to <2>diallabel;

send 'Z-TRANS' to <3>diallabel;
send 'Z-TRANS' to <3>diallabel;
send 'INT' to <3>diallabel;
send 'INT' to <3>diallabel;
send 'INT' to <3>diallabel;
send 'Z-TRANS' to <3>diallabel;
send 'Z-TRANS' to <3>diallabel;
send 'Z-TRANS' to <3>diallabel;

send 'SCALE' to <4>diallabel;
send 'X-ROT' to <5>diallabel;
send 'Y-ROT' to <6>diallabel;
send 'Z-ROT' to <7>diallabel;
send 'BACKCLIP' to <8>diallabel;

way := f:sync(2);
connect way<2>:<1>dialmux;
connect way<2>:<1>dialmux2;
connect way<2>:<2>way;
connect way<2>:<1>r;
connect way<2>:<9>waylabel;
connect way<2>:<2>bits;
connect way<2>:<9>diallabel;
send fix(1) to <2>way;
send fix(2) to <2>way;
send fix(3) to <2>way;
send fix(4) to <2>way;
send fix(5) to <2>way;
send fix(6) to <2>way;
send fix(7) to <2>way;
send fix(8) to <2>way;
send TRUE to <1>way;  {activate it}
```

```
{network to change from solid to surface}

sslabel := f:boolean_choose;
connect sslabel<1>:<1>flabel7;
send 'SOLID' to <2>sslabel;
send 'SURFACE'  to <3>sslabel;

issolid := f:nop;
connect issolid<1>:<2>world.rendering;
connect issolid<1>:<1>sslabel;
ss := f:sync(2);
connect ss<2>:<2>ss;
connect ss<2>:<1>issolid;
send TRUE to <2>ss;
send FALSE to <2>ss;
send FALSE to <1>ss; { initially a surface }

{network to toggle anti-alias possibilities}

aalabel := f:sync(2);
aavalue := f:sync(2);
connect aavalue<1>:<5>shadingenvironment;
send 'NO-AA ' to <1>aalabel;
send 'EDGEAA' to <1>aalabel;
send 'FULLAA' to <1>aalabel;
send fix(0) to <1>aavalue;
send fix(1) to <1>aavalue;
send fix(2) to <1>aavalue;
connect aavalue<1>:<1>aavalue;
connect aalabel<1>:<1>aalabel;
connect aalabel<1>:<1>flabel5;
send fix(0) to <2>aalabel;
send fix(0) to <2>aavalue;
{network to control rendering style}

stylab := f:sync(2);
styval := f:sync(2);
style := f:const;
send 'HIDDEN' to <1>stylab;
send 'WASH' to <1>stylab;
send 'FLAT' to <1>stylab;
send 'PHONG ' to <1>stylab;
send 'GOURAUD' to <1>stylab;
send 'XSECTION' to <1>stylab;
send 'SECTION' to <1>stylab;
send 'BACKFACE' to <1>stylab;
send 'SAVE-SEC' to <1>stylab;
send 'SAVE-HL' to <1>stylab;
send fix(4) to <1>styval;
send fix(5) to <1>styval;
send fix(6) to <1>styval;
send fix(7) to <1>styval;
send fix(8) to <1>styval;
send fix(1) to <1>styval;
send fix(2) to <1>styval;
send fix(3) to <1>styval;
send 'object' to <1>styval;
send 'hlview' to <1>styval;
conn stylab<1>:<1>stylab;
conn stylab<1>:<1>flabel3;
conn styval<1>:<1>styval;
conn styval<1>:<2>style;

send fix(0) to <2>styval;
send fix(0) to <2>stylab;

{ some useful viewports }

piclab := f:sync(2);
picval := f:sync(2);
send 'SQUARE' to <1>piclab;
send 'BIG-PIC' to <1>piclab;
send '1-OF-2' to <1>piclab;
send '2-OF-2' to <1>piclab;
send '1-OF-6' to <1>piclab;
send '2-OF-6' to <1>piclab;
send '3-OF-6' to <1>piclab;
send '4-OF-6' to <1>piclab;
send '5-OF-6' to <1>piclab;
send '6-OF-6' to <1>piclab;
send v3d (80,0,479) to <1>picval;
send v3d (0,-80,639) to <1>picval;
send v3d (0,80,319) to <1>picval;
send v3d (320,80,319) to <1>picval;
send v3d (5,240,209) to <1>picval;
send v3d (215,240,209) to <1>picval;
send v3d (425,240,209) to <1>picval;
send v3d (5,30,209) to <1>picval;
send v3d (215,30,209) to <1>picval;
send v3d (425,30,209) to <1>picval;

conn piclab<1>:<1>piclab;
conn piclab<1>:<1>flabel2;
conn picval<1>:<1>picval;
conn picval<1>:<3>shadingenvironment;
conn picval<1>:<4>tripcolor;

send 1 to <2>piclab;
send 1 to <2>picval;
```

```
{network to invert overlay-picture mechanism}

overinvert := f:xorc;
send true to <2>overinvert;
overlabel := f:sync(2);
connect overinvert<1>:<9>shadingenvironment;
connect overinvert<1>:<2>overlabel;
send 'REFRESH' to <1>overlabel;
send 'OVERLAY' to <1>overlabel;
connect overlabel<1>:<1>overlabel;
connect overlabel<1>:<1>flabel4;

{network to invert color interpolation}

colorinvert := f:xorc;
send true to <2>colorinvert;
colorlabel := f:sync(2);
connect colorinvert<1>:<10>shadingenvironment;
connect colorinvert<1>:<2>colorlabel;
send 'COL-OFF' to <1>colorlabel;
send 'COL-ON ' to <1>colorlabel;
connect colorlabel<1>:<1>colorlabel;
connect colorlabel<1>:<1>flabel8;

{ ********** }
primsync := f:sync(3); { this sync to ensure that lines and spheres have }
                       { been xformed before rendering is actually       }
                       { triggered                                       }

connect style<1>:<3>primsync; { style is where polygon rendering type is }
                              { stored                                   }
connect primsync<3>:<1>world.rendering;
conn primsync<1>:<3>world.rendering;  { connect lines up }
conn primsync<2>:<4>world.rendering;  { connect spheres up }

conn linexformdata<1>:<1>primsync; { connect lines to sync function }
conn linexformdata<1>:<1>spherexformdata; { Let line xformdata trigger }
                                          { sphere xformdata }
conn spherexformdata<1>:<2>primsync; { connect spheres to sync function }
{ ********** }

{ buttons }
fkmo := f:switch;
connect fkeys<1>:<1>fkmo;

{ ********** }
connect fkmo<1>:<1>linexformdata;    { Trigger line xformdata }
connect fkmo<1>:<1>style;            { Trigger rendering style constant }
{ ********** }
connect fkmo<2>:<2>piclab; connect fkmo<2>:<2>picval;
connect fkmo<3>:<2>stylab; connect fkmo<3>:<2>styval;
connect fkmo<4>:<1>overinvert;
connect fkmo<5>:<2>aavalue;
connect fkmo<5>:<2>aalabel;
connect fkmo<6>:<7>shadingenvironment;
connect fkmo<7>:<1>ss;
connect fkmo<8>:<1>colorinvert;
connect fkmo<9>:<8>shadingenvironment;
connect fkmo<10>:<2>r;
connect fkmo<10>:<1>original;
connect fkmo<11>:<1>bits;
connect fkmo<11>:<1>original;
connect fkmo<12>:<1>way;

fkm := f:inputs_choose(13);
connect fkm<1>:<2>fkmo;
connect fkeys<1>:<13>fkm;
send fix(1) to <1>fkm;
send fix(2) to <2>fkm;
send fix(3) to <3>fkm;
connect overinvert<1>:<4>fkm;
send fix(0) to <5>fkm;
send true  to <6>fkm;
send fix(7) to <7>fkm;
connect colorinvert<1>:<8>fkm;
send 'ABORT' to <9>fkm;
send v3d(0,0,0) to <10>fkm;
send fix(11) to <11>fkm;
send fix(12) to <12>fkm;

send 'RENDER' to <1>flabel1;
send 'CLEAR' to <1>flabel6;
send 'ABORT'    to <1>flabel9;
send 'RESET' to <1>flabel10;
send 'ON/OFF' to <1>flabel11;
send true to <1>overinvert;
send true to <1>colorinvert;

{ some useful colors }

blue      := attr color 0;
magenta   := attr color 60;
purple    := attr color 90;
red       := attr color 120;
orange    := attr color 150;
yellow    := attr color 180;
green     := attr color 240;
turquoise := attr color 270;
cyan      := attr color 300;
white     := attr color 0, 0, 1;
grey      := attr color 0, 0, 0.5;
black     := attr color 0, 0, 0;
```

```
{some other names for shadingenvironment}

se := f:pass(14);
connect se<1>:<1>shadingenvironment;
connect se<2>:<2>shadingenvironment;
connect se<3>:<3>shadingenvironment;
connect se<4>:<4>shadingenvironment;
connect se<5>:<5>shadingenvironment;
connect se<6>:<6>shadingenvironment;
connect se<7>:<7>shadingenvironment;
connect se<8>:<8>shadingenvironment;
connect se<9>:<9>shadingenvironment;
connect se<10>:<10>shadingenvironment;
connect se<11>:<11>shadingenvironment;
connect se<12>:<12>shadingenvironment;
connect se<13>:<13>shadingenvironment;
connect se<14>:<14>shadingenvironment;
ambient := f:pass(1);
connect ambient<1>:<1>shadingenvironment;
background := f:pass(1);
connect background<1>:<2>shadingenvironment;
rasterviewport := f:pass(1);
connect rasterviewport<1>:<3>shadingenvironment;
exposure := f:pass(1);
connect exposure<1>:<4>shadingenvironment;
anti-alias := f:pass(1);
connect anti-alias<1>:<5>shadingenvironment;
depth := f:pass(1);
connect depth<1>:<6>shadingenvironment;
screenwash := f:pass(1);
connect screenwash<1>:<7>shadingenvironment;

{ make PS300 come up in shift line/local }
send 'R' to <1>kbhandler;

{EOF}
```



**Display Tree and Function Network for Displaying Spheres and Lines**

# CHANGE PAGES FOR THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND THE GRAPHICS SUPPORT ROUTINE MANUALS

# ATTRIBUTES

RENDERING - Data Structuring (PS 340)

Version A2.V01

## FORMAT

name := ATTRIBUTES **attributes** [AND **attributes**];

## DESCRIPTION

Specifies the various characteristics of polygons used in the creation of shaded renderings. This command is only used with the PS 340. For a detailed explanation of defining and interacting with shaded images, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2B.

## PARAMETERS

**attributes** - The attributes of a polygon are defined as follows.

[COLOR **h**[,**s**[,**i**]]] [DIFFUSE **d**] [SPECULAR **s**] [OPAQUE **t**]

where

**h** - is a real number specifying the hue in degrees around the color wheel. Pure blue is 0 and 360, pure red is 120, and pure green is 240.

**s** - is a real number specifying saturation. No saturation (gray) is 0 and full saturation (full toned colors) is 1.

**i** - is a real number specifying intensity. No intensity (black) is 0, full intensity (white) is 1.

**d** - is a real number from 0 to 1 specifying the proportion of color contributed by diffuse reflection versus that contributed by specular reflection. Increasing **d** makes the surface more matte. Decreasing **d** makes it more shiny.

**s** - is an integer from 0 to 255 which adjusts the concentration of specular highlights. The more metallic an object is, the more concentrated the specular highlights.

**t** - is a real number from 0 to 1 specifying the transparency of the polygon, with 1 being fully opaque and 0 being fully transparent (invisible).

## DEFAULTS

If no color is specified, the default is white (s = 0, i = 1). If saturation and intensity are not specified, they default to 1. If only hue and saturation are specified, intensity defaults to 1. If no diffuse attribute is given, d defaults to .75. If no specular attribute is given, s defaults to 4. If no opaque attribute is given, the default is 1 (fully opaque).

## NOTES

1. Polygon attribute nodes are created in mass memory but are not part of a display tree. The attributes specified in an ATTRIBUTES command are assigned to polygons which include a WITH ATTRIBUTES clause. The attributes specified in a WITH ATTRIBUTES clause of a POLYGON command apply to all subsequent polygons until superseded by another WITH ATTRIBUTES clause. If no WITH ATTRIBUTES option is given for a POLYGON node, default attributes are assumed. The default attributes are 0,0,1 for COLOR, 0.75 for DIFFUSE, and 4 for SPECULAR.

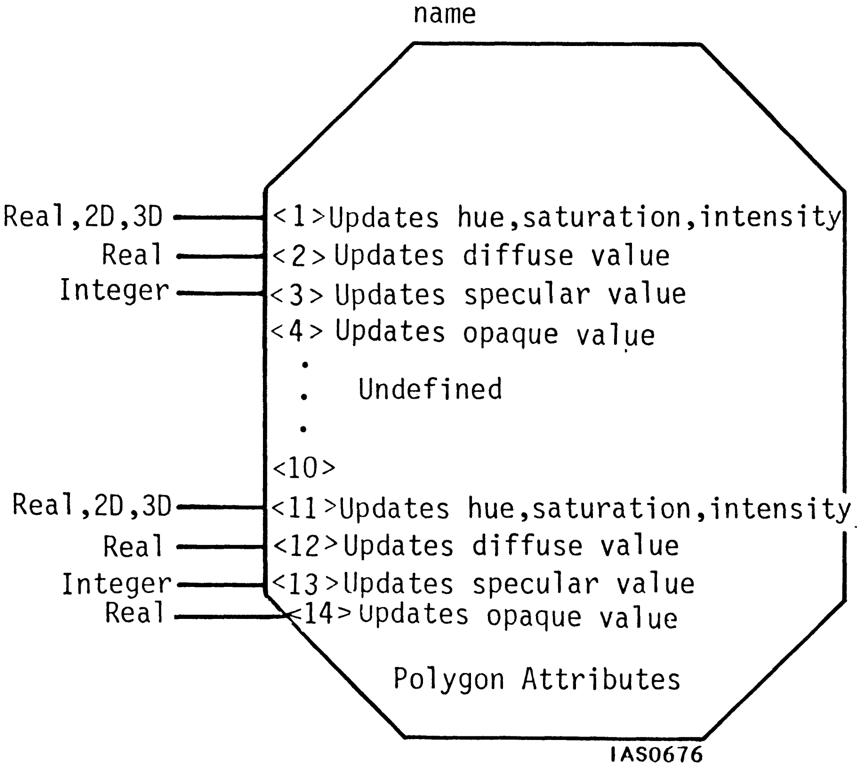
2. The various attributes may be changed from a function network via inputs to an attribute node, but the changes have no effect until a new rendering is created.

3. A second set of attributes may be given after the word AND in the ATTRIBUTES command. These attributes apply to the obverse side of the polygon(s) concerned. In other words, the two sides of an object may have different attributes. The attributes defined in the first attributes pertain to front-facing polygons. Those in the AND attributes clause pertain to backfacing polygons.

## NODE CREATED

Polygon ATRRIBUTES definition node. This node resides in mass memory, but is not included in a display tree.

INPUTS FOR UPDATING NODE


name
Real,2D,3D
<1>Updates hue,saturation,intensity
Real
<2> Updates diffuse value
Integer
<3> Updates specular value
<4> Updates opaque value
Undefined
<10>
Real,2D,3D
<11>Updates hue,saturation,intensity
Real
<12>Updates diffuse value
Integer
<13>Updates specular value
Real
<14> Updates opaque value
Polygon Attributes
IAS0676

NOTES ON INPUTS

1. Inputs <1> and <11> accept a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation and intensity.

2. Values sent to inputs <1>, <2>, and <3> specify the color and attributes for shading the front of the polygon(s) or for both sides if no obverse attributes are given. (Values sent to inputs <11>, <12>, and <13> specify the color and attributes for shading the obverse side of the polygon.

3. Inputs <4> and <14> accept a real number to update the opaque value of the polygon's attributes.

4. If anything other than a 3D vector is sent to input <1> or <11>, default values for the other variables are assumed.

Version A2.V01

## FORMAT

```
name := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [COPLANAR]
        POLYGon vertex ... vertex;
```

## DESCRIPTION

Allows you to define primitives as solids and surfaces. This command is only used with the PS 340. For a detailed explanation of defining and interacting with polygons, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2B.

## PARAMETERS

**WITH ATTRIBUTES** – An option that assigns the attributes defined by **name1** for all polygons until superseded by another WITH ATTRIBUTES clause.

**WITH OUTLINE** – An option that specifies the color of the edges of a polygon on the color CSM display, or their intensity on a black and white display as a real number (**h**).

**COPLANAR** – Declares that the specified polygon and the one immediately preceding it have the same plane equation. This should only be used to define holes in the interior of a polygon.

**vertex** – A vertex is defined as follows:

```
[ S ] x,y,z [ N x,y,z ] [C h[,s[i]]]
```

where

S – indicates that the edge drawn between the previous vertex and this one represents a soft edge of the polygon. If the S specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft. This option is only necessary to use if soft edges are desired in hidden-line pictures on the calligraphic display. This option has no effect on renderings displayed on the raster screen.

PARAMETERS (continued)

N - Indicates a normal to the surface with each vertex of the polygon. Normals are used only in smooth-shaded renderings. Normals must be specified for all vertices of a polygon or for none of them. If no normals are given for a polygon, they are defaulted to the same as the plane equation for the polygon.

x, y, z - are coordinates in a left-handed Cartesian system.

C - indicates a color to be assigned to the vertex. This color will be interpolated across the polygon to the other vertices. Color must be specified for all vertices of a polygon or for none of them (4K ACP only).

h, s, i - are coordinates of the Hue-Saturation-Intensity color system.

Polygons may be solidly colored by specifying a color through the attributes command or the colors may be assigned to the vertices by giving a color with each vertex specified. The color is specified by giving first the vertex and then the color (h, s, i). If just the hue and saturation are given, the intensity will default to 1. If just the hue value is given, the saturation and intensity will default to 1. If no vertex colors are given, the vertex colors will default to those specified in the ATTRIBUTE clause.

Vertex colors must be specified for all vertices of a polygon or for none of them. However, as with normals, some polygons may have color at their vertices while others polygons do not have color at their vertices. This means that it is possible to have some objects in the picture color interpolated, while others are not.

Although color of polygon vertices is specified h, s, i, the colors are linearly interpolated across the vertices in the Red-Green-Blue color system. If colors are not interpolating the way you would like them to, add more vertices to the polygon, or break up large solid volumes into smaller sub-volumes and assign the desired colors to the new vertices in the object.

You can specify color for a polygon with both the ATTRIBUTES command and the color by vertex specification. A new input to the SHADINGENVIRONMENT function allows you to switch between attribute-defined color and vertex-defined color. Input <10> of SHADINGENVIRONMENT accepts a Boolean to determine how color will be specified. To use vertex colors rather than surface attributes, send TRUE to input <10> of SHADINGENVIRONMENT. To return to using the attributes specified in the ATTRIBUTE command, send FALSE to input <10> of SHADINGENVIRONMENT.

NOTES

1. A polygon declared to be coplanar must lie in the same plane as the previous polygon if correct renderings are to be obtained. The system does not check for this condition. Coplanar polygons may be defined without the COPLANAR specifier, unless outer and inner contours are being associated.

2. To use the COPLANAR specifier to define a hole, the vertices of the hole must be ordered in a counter-clockwise direction, while the vertices of the surrounding polygon must be ordered in a clockwise direction.

3. All members of a set of consecutive coplanar polygons are taken to have the same plane equation, that of the previous polygon not containing the coplanar option. If COPLANAR is specified for the first polygon in a node, it has no effect.

4. If the **N** (normal) specifier is specified for a vertex in a polygon, it must be specified for all vertices in that polygon. The same is true for the C (color at vertex) specifier.

5. If the S (soft) specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft.

6. No more than 250 vertices per POLYGon may be specified.

7. The last defined vertex in the polygon is assumed to connect to the first defined vertex; that is, polygons are implicitly closed.

8. There is no syntactical limit for the number of POLYGon clauses in a group.

9. The ordering of vertices within each POLYGon has important consequences for rendering operations.

DISPLAY TREE NODE CREATED

POLYGON data node.

INPUTS FOR UPDATING NODE

None.

Version A2.V01

## FORMAT

name := SOLID_rendering APPLied to name1;

## DESCRIPTION

Declares a POLYGon object to be a solid and marks the object so that rendering operations can be performed on it. This command creates a rendering node. It is used exclusively with the PS 340.

## PARAMETERS

**name1** – Either a POLYGon node or an ancestor of one or more POLYGon nodes.

## NOTES

1. If non-POLYGon data nodes (VECtor/list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in **name1**, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes. However, special vector lists output from F:XFORMDATA used to display spheres and lines on the raster display can be used and will be displayed if rendered.

2. IF and SET Conditional_BIT, IF and SET LEVel_of_detail, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operation nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result.

NOTES (continued)

Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings. The transformation nodes mentioned above may also be placed above the rendering node.

4. The five nodes WINDOW, VIEWport, EYE, Field_Of_View, and Matrix_4X4 should not, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWport, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiply-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR, SET PLOTTER) not mentioned above, are ignored by rendering operations when placed beneath a rendering node. Such nodes must therefore be placed above a rendering node if they are to have their customary effects on renderings. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPlayed. If the object itself is DISPlayed but its rendering node is not, no renderings can be created.

8. Any input to input<1> of a rendering node causes an output. Inputs sent to input<2> will not cause an output to be sent. If output<1> has not been connected, and an integer, string, or Boolean is sent to input<1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

NOTES (continued)

9. Input <3> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering.

10. Input <4> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets each vector as an x,y,z spherical

11. Input <5> of the rendering node accepts the name of the original vector list (sent to F:XFORMDATA with its output <1> sent to input <4> of the rendering node) to enable accurate scaling for rendering raster lines and spheres.

12. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_rendering or SURFACE_rendering node) works only after requesting hidden-line pictures, backface pictures, sectioned pictures, or cross-sectioned pictures. Raster images may not be toggled.

13. Sending a fix(7) to input <1> of the SOLID_rendering or SURFACE_rendering node produces a type of Phong shading. Phong shading is made by interpolating the surface normal between vertices of the polygon and then calculating the correct lighting at each pixel. This is the highest quality of smooth shading currently supported.

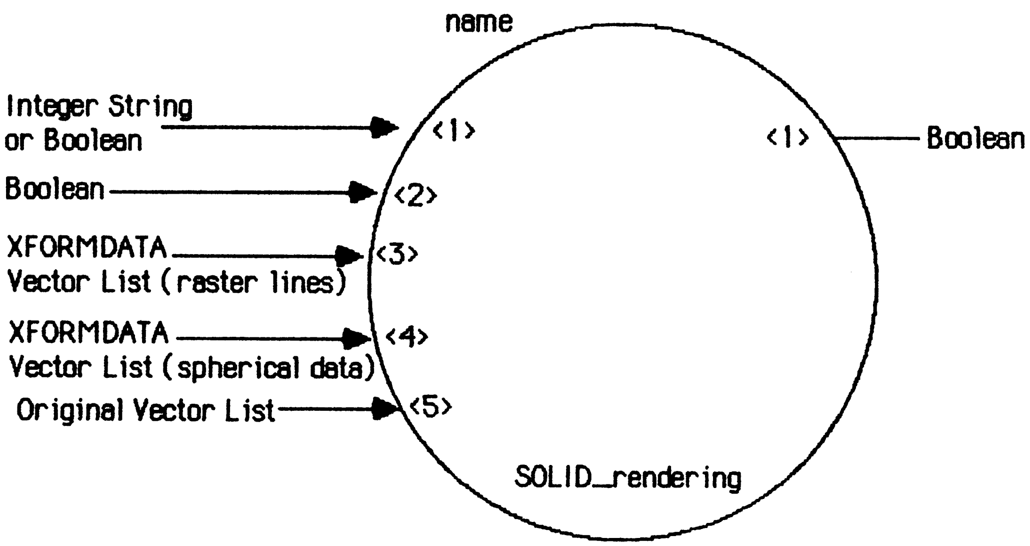
14. Sending a fix(8) to input <1> of the SOLID_rendering or SURFACE_rendering node will produce a type of Gouraud shading. Gouraud shading is made by calculating the correct lighting at the vertices of the polygon only and interpolating the intensity across the polygon to produce a smooth-shaded picture. An image produced with Gouraud shading will not be the quality of an image produced with Phong shading, but the Gouraud-shaded image will be produced at a faster rate. The user must supply normals at each of the polygons for the object to be smooth-shaded.

15. Sending data a non-existent rendering node input, will cause the system to crash.

DISPLAY TREE NODE CREATED

Rendering operation node.

## INPUTS FOR UPDATING NODE


name
Integer String or Boolean
<1>
Boolean
<2>
XFORMDATA Vector List (raster lines)
<3>
XFORMDATA Vector List (spherical data)
<4>
Original Vector List
<5>
<1>
Boolean
SOLID_rendering

## NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object on the calligraphic display.
1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only).
2: Creates and displays a sectioned rendering on the calligraphic display.
3: Creates and displays a rendering using backface removal (solid only) on the calligraphic display.
4: Creates and displays a rendering using hidden-line removal on the calligraphic display.
5: Generates a wash-shaded image on the raster display.
6: Generates a flat-shaded image on the raster display.
7: Generates a Phong-shaded image on the raster display.
8: Generates a Gouraud-shaded image on the raster display.

NOTES ON INPUTS (continued)

String: Causes the current rendering to be saved under the name given in the string (calligraphic display only).

False: Sets the original view. The original descendent structure of the rendering operate node is displayed.

True: Sets the rendered view. The rendered view of the original descendent structure of the rendering operate node.

Input <2>

True: Declares the object to be a solid.

False: Declares the object to be a surface.

Input <3>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define raster lines.

Input <4>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define spherical centers.

Input <5>

Accepts the original vector list to enable accurate spherical scaling.

Output <1>

True: Rendering is displayed

False: Rendering is not displayed

## FORMAT

name := SURFACE_rendering APPLied to **name1**;

## DESCRIPTION

Declares a POLYGon object to be a surface and marks the object so that rendering operations can be performed on it. This command creates a rendering node. It is used exclusively with the PS 340.

## PARAMETERS

**name1** - Either a POLYGon node or an ancestor of one or more POLYGon nodes.

## NOTES

1. If non-POLYGon data nodes (such as VECtor_list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in **name1**, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes. However, special vector lists output from F:XFORMDATA used to display spheres and lines on the raster display can be used and will be displayed if rendered.

2. IF and SET conditional_BIT, IF and SET LEVel_of_detail, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operation nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result.

NOTES (continued)

Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings. The transformation nodes mentioned above may also be placed above the rendering node.

4. The five nodes WINDOW, VIEWport, EYE, Field_Of_View, and Matrix_4X4 should not, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWport, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiply-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR, SET PLOTTER) not mentioned above, are ignored by rendering operations when placed beneath a rendering node. Such nodes must therefore be placed above a rendering node if they are to have their customary effects on renderings. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPlayed. If the object itself is DISPlayed but its rendering node is not, no renderings can be created.

8. Any input to input<1> of a rendering node causes an output. Inputs sent to input<2> will not cause an output to be sent. If output<1> has not been connected, and an integer, string, or Boolean is sent to input<1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

NOTES (continued)

9. Input <3> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering.

10. Input <4> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets each vector as an x,y,z spherical

11. Input <5> of the rendering node accepts the name of the original vector list (sent to F:XFORMDATA with its output <1> sent to input <4> of the rendering node) to enable accurate scaling for rendering lines and spheres.

12. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_rendering or SURFACE_rendering node) works only after requesting hidden-line pictures, backface pictures, sectioned pictures, or cross-sectioned pictures. Raster images may not be toggled.

13. Sending a fix(7) to input <1> of the SOLID_rendering or SURFACE_rendering node produces a type of Phong shading. Phong shading is made by interpolating the surface normal between vertices of the polygon and then calculating the correct lighting at each pixel. This is the highest quality of smooth shading currently supported.

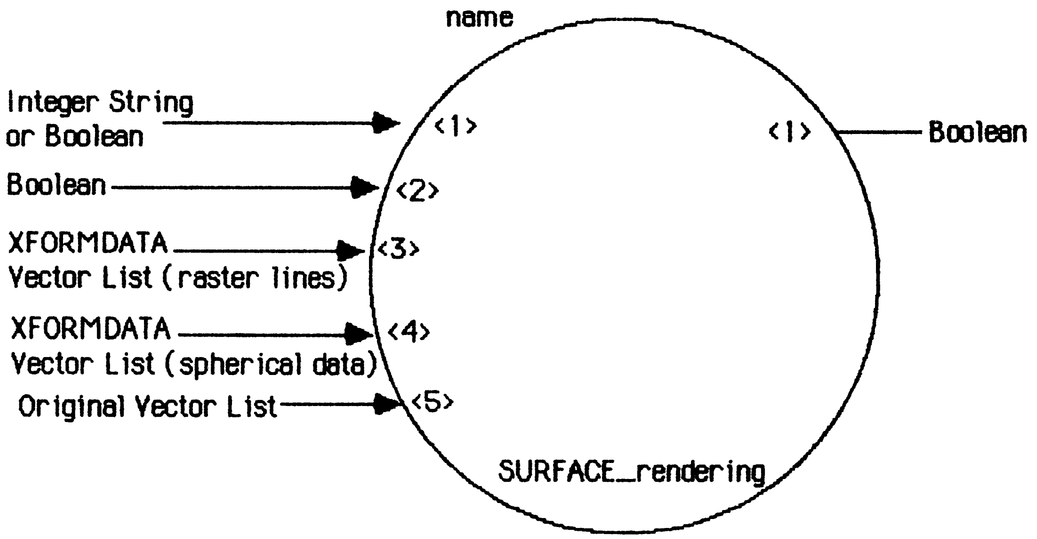
14. Sending a fix(8) to input <1> of the SOLID_rendering or SURFACE_rendering node will produce a type of Gouraud shading. Gouraud shading is made by calculating the correct lighting at the vertices of the polygon only and interpolating the intensity across the polygon to produce a smooth-shaded picture. An image produced with Gouraud shading will not be the quality of an image produced with Phong shading, but the Gouraud-shaded image will be produced at a faster rate. The user must supply normals at each of the polygons for the object to be smooth-shaded.

15. Sending data a non-existent rendering node input, will cause the system to crash.

## DISPLAY TREE NODE CREATED

Rendering operation node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object on the calligraphic display.

1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only) on the calligraphic display.

2: Creates and displays a sectioned rendering on the calligraphic display.

3: Creates and displays a rendering using backface removal (solid only) on the calligraphic display.

4: Creates and displays a rendering using hidden-line removal on the calligraphic display.

5: Generates a wash-shaded image on the raster display.

6: Generates a flat-shaded image on the raster display.

7: Generates a Phong-shaded image on the raster display.

8: Generates a Gouraud-shaded image on the raster display.

NOTES ON INPUTS (continued)

String: Causes the current rendering to be saved under the name given in the string (calligraphic display only).

False: Sets the original view. The original descendant structure of the rendering operate node is displayed.

True: Sets the rendered view. The rendered view of the original descendent structure of the rendering operate node.

Input <2>

True: Declares the object to be a solid.

False: Declares the object to be a surface.

Input <3>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define raster lines.

Input <4>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define a spherical center.

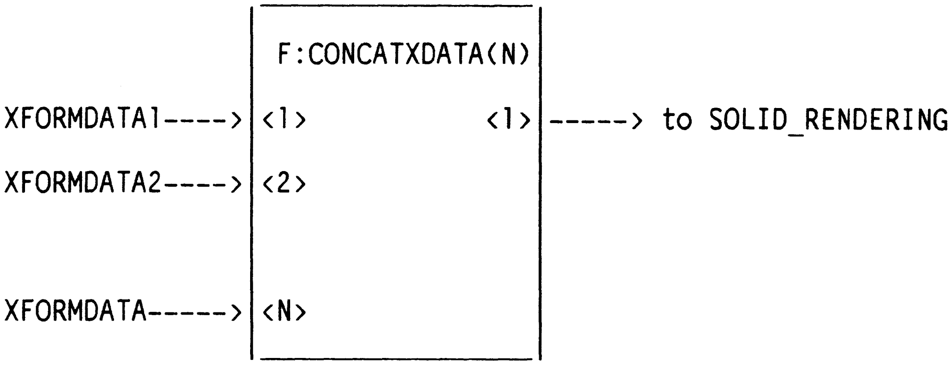
Input <5>

Accepts the original vector list to enable accurate spherical scaling.

Output <1>

True: Rendering is displayed

False: Rendering is not displayed



## PURPOSE

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

## DESCRIPTION

INPUT

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

.

.

.

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

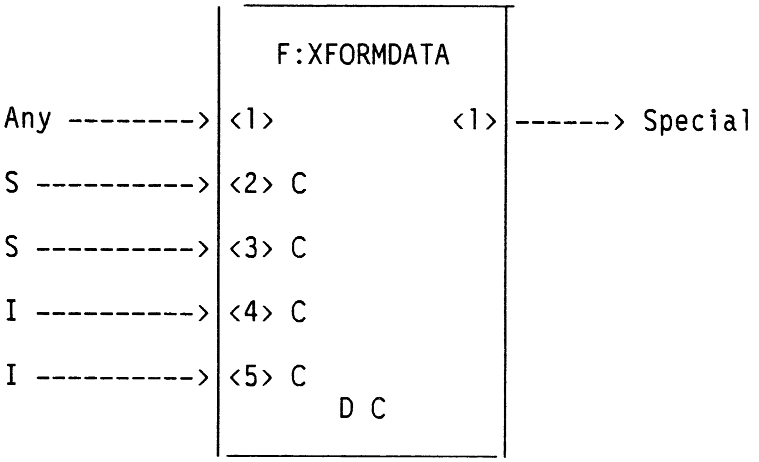
OUTPUT

<1> - concatenated vector list

NOTES

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

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



## PURPOSE

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

## DESCRIPTION

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

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

## DEFAULTS

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

NOTES

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

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

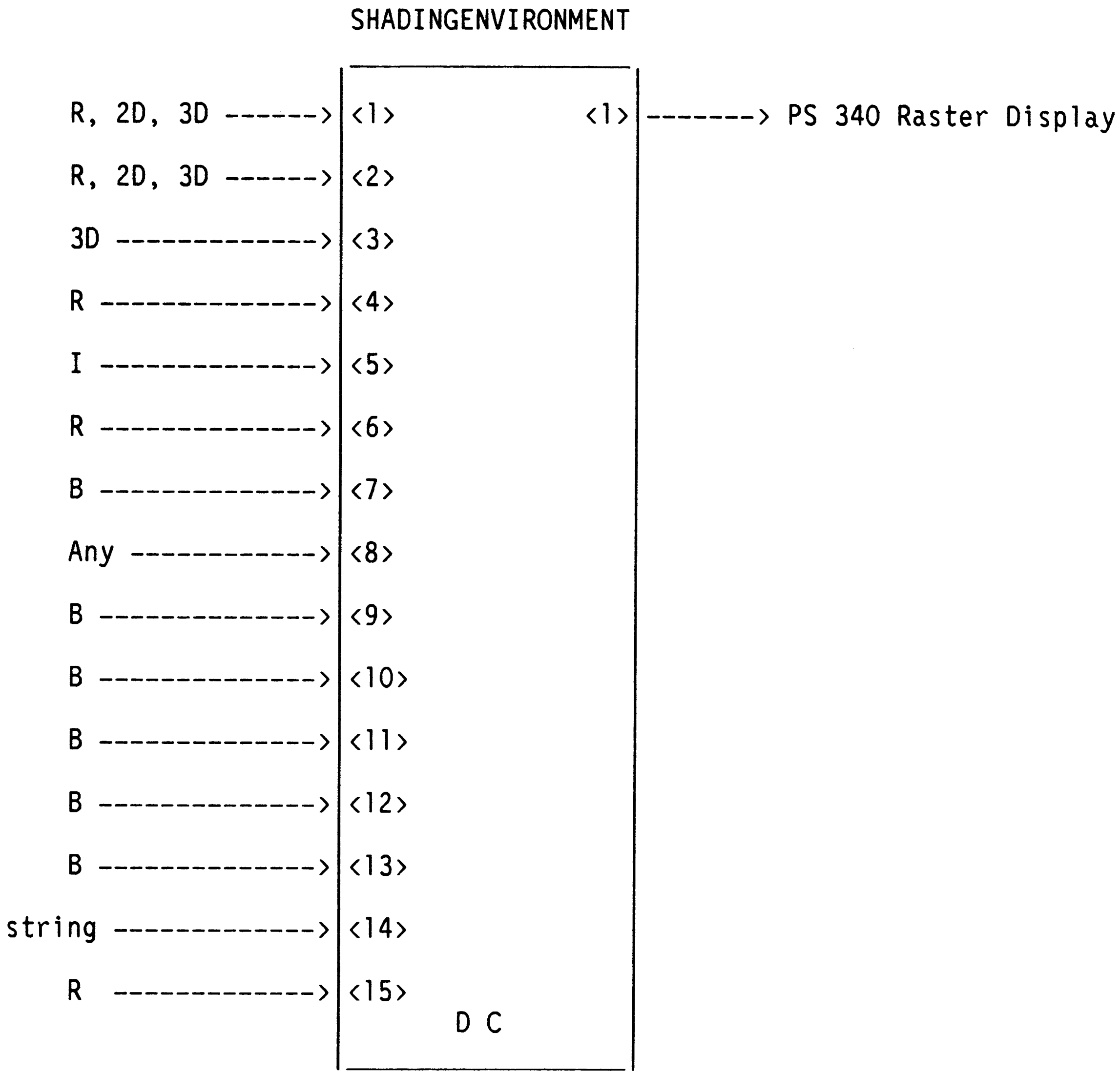
3. Input <3> is a string containing the name to be associated with the transformed vectors. The name need not be previously defined. If this input does not contain a valid string, the transformed matrix or vectors will be created without a name (an acceptable situation unless the transformed vectors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

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

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

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

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



## PURPOSE

For use with the PS 340 system, this function allows you to control various non-dynamic factors of shaded renderings displayed on the raster screen.

## DESCRIPTION

INPUT

<1> - ambient color
<2> - background color
<3> - raster viewport
<4> - exposure
<5> - edge-smoothing (anti-aliasing) control
<6> - depth cueing
<7> - screen wash
<8> - user abort
<9> - overlay/refresh control
<10> - color by vertex control
<11> - opaque (transparency) control
<12> - specular highlights control
<13> - special color blending for spheres control
<14> - update attribute table
<15> - line z-value control

OUTPUT

<1> - connected to the PS 340 Raster Display

## NOTES

1. Ambient color: input <1> accepts a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation, and intensity, to specify the ambient color. The ambient color is combined with the result obtained from the light sources to determine the color of ambient light. The default ambient color is white, with a default intensity of .25. The ambient color is analogous to the color reflected off a wall.

2. Background color: input <2> accepts a real number as hue, a 2D vector as hue and saturation, and or a 3D vector as hue, saturation, and intensity to specify the background color. The raster screen will be colored with the background color prior to any shaded rendering done in the refresh mode (refer to input <9>). The default background color is black (0,0,0).

3. Raster viewport: input <3> accepts a 3D vector as the viewport on the raster image buffer where shaded renderings will be displayed. Raster viewports are always square, the lower left corner being given by the X and Y coordinates of the vector, and its size given by the Z coordinate, such that the upper right corner is at (x+z,y+z). Values are rounded to the nearest pixel. The default viewport is (80,0,480). The viewport is not intended for magnification of small parts of the calligraphic image, but for mapping the square vector display onto the rectangular raster display.

NOTES

The viewport is also intended to allow multiple images to be generated side by side on the raster display. Thus, the largest recommended value for the viewport is (0,-80,640). The actual largest viewport is somewhat larger and depends on combinations of the three values. The image is clipped to the physical raster for which $0 \le X < 640$ and $0 \le Y < 480$.

4. Exposure: input <4> accepts a real number as the exposure, controlling the overall brightness of the picture. The exposure is like that on a camera. If a picture is taken of an object with a very bright specular highlight, it may be so bright that the rest of the object is darkened. If three light sources exist, the object would be about three times brighter, making the object too bright. The exposure should be brought down to control this.

   The exposure is multiplied by the intensity at each pixel and the result clipped to the maximum intensity. This enables the overall brightness of a rendering to be increased without causing bright spots to exceed maximum intensity (instead forming "plateaus" of maximum intensity). Note that this may cause changes in color on a plateau, where color has reached its maximum, but the others have not. Exposure values may vary between .3 and 3, values outside that range being changed to .3 or 3. The default exposure is 1.

5. Edge smoothing (anti-aliasing) control: input <5> accepts an integer which will allows users to choose between having a relatively fast rendering with jagged edges along the edges of polygons and having a slower rendering with smoother edges. Anti-aliasing is accomplished by taking 16 samples per pixel instead of just one. You are given the choice of having no edge smoothing at all, smoothing along the edges only, or sampling 16 times within every pixel for every polygon.

   Sending fix(0) to this input produces no smooth edges, and is fastest. The polygons are rendered with one sample per pixel.

   Sending fix(1) produces smooth edges, but may not correctly resolve obscurity between surfaces that are extremely close in z-values or that are interpenetrating. The 16 samples are taken only where the edges of the polygon touch a pixel. The interior of the polygons are still rendered with one sample per pixel. This has a speed intermediate between a fix(0) and a fix(2).

   Sending fix(2) to input <5> of SHADINGENVIRONMENT produces edge-smoothing, and is slower, but resolves surfaces. The 16 samples are taken in every pixel of every polygon. The default is 0.

NOTES (continued)

6. Depth cueing: input <6> accepts a real number in the range of 0 to 1 to control the effect of depth cueing in the shaded image (1 specifying no depth cueing and 0 specifying maximum depth cueing). As perceived depth from the viewer increases, the colors are mixed with the ambient light color (input <1> of SHADINGENVIRONMENT). Thus, if a 3D vector(0,0,0) (black) is sent to the ambient input <1> and if a 0 is sent to input <6>, the objects will be rendered with a ramp ending in black at the back clipping plane. A 1 sent to input <6> will turn off depth cueing. The default is 0.2 giving a fairly large depth cueing effect.

7. Screen wash: input <7> accepts a Boolean, and is the only input to cause a visual effect immediately. TRUE causes the whole physical raster screen to be filled with the current background color, while FALSE just fills the currently defined viewport (clipped to the screen).

8. User abort: input <8> accepts any message and causes abort to occur for the current hidden-line rendering on the calligraphic screen. Sometimes the hidden-line algorithm can take a long time to run to completion. This input allows you to exit rendering before it is complete.

9. Clear/Overlay Control: input <9> accepts a Boolean which determines whether the screen is to be cleared with the current background color before the rendering is done. Sending a TRUE to this input causes the current object to be rendered on top of the image currently being displayed on the raster screen. Sending a FALSE to this input causes the screen to be washed clean with the current background color. The object which overlays the image is is anti-aliased with the background color so the object is correctly composited into the image with no jagged edges. The default is false.

10. Color by Vertex Control: input <10> accepts a Boolean which turns off (or on) the use of vertex colors. Color by vertex is accomplished by defining a color for each vertex in the polygon:

    ```
    [S] x,y,z  [N x,y,z]  [C h [,s [,i] ] ]
    ```

    Refer to the notes on the POLYGon command for more information about how to add color by vertex. To use the vertex colors defined this way rather than the color defined in the ATTRIBUTES, send TRUE to this input. Send FALSE to this input to return to using the color specified by the ATTRIBUTES command. The default is false.

11. Opaque (Transparency) Control: input <11> accepts a Boolean which allows you to turn off (or on) the transparency assigned to the polygon with the OPAQUE clause of the attribute command. Transparent polygons are created by modifying the ATTRIBUTE command in the following manner.

NOTES (continued)

```
Name := ATTRIBUTE  [Color h [,s [,i] ] ] [OPAQUE t]
                   [Diffuse d] [Specular s];
```

where t refers to a value between 0 and 1, with 1 being a fully opaque polygon and 0 being a fully transparent polygon. The default is false.

As t decreases from 1 to 0, more of the color of the obscured object(s) will show through. At t=0, the transparent polygon becomes completely invisible. If no opaque attribute is specified, the default is 1 (fully opaque).

12. Specular Highlight Control: input <12> accepts a Boolean which allows you to turn on (or off) specular highlights. Flat, Gouraud, and Phong shading all use the same shading equation. This means that multiple light sources are processed in each case and that specular highlights are calculated. Specular highlights may appear strange in Gouraud or flat shading. In Gouraud shading, the highlights may cause bright horizontal bands to appear inside the polygons. In flat shading, some polygons will appear very bright when viewed from certain angles unless specular highlights are turned off. The default is true.

13. Special Color Blending for Spheres: input <13> accepts a Boolean which turns off (or on) the color blending used for correct spherical rendering. Sending a TRUE to input <13> turns ON this special color blending. Sending a FALSE to input <13> turns OFF special color blending. The default is false.

14. Update Attribute Table: input <14> accepts a 3D tabulated vector list to update the attribute table that specifies color, radii, diffuse, and specular highlights for spheres and lines. The attribute table has 0 to 127 entries with six table components for each entry. The attribute table can be updated by encoding the table entries into a PS 300 vector list and then sending the name of the vector list to input <14>. The six table components are encoded into two consecutive 3D tabulated vectors of the vector list. Hue, saturation, and intensity are encoded into the first x,y,z respectively. Radius, diffuse, and specular are encoded into the second x,y,z respectively. The table index is encoded into the t field of the second vector.

The table is initialized as follows:

| INDEX | Hue | Sat | Intensity | Radius | Diffuse | Specular |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0.5 | 1.8 | 0.7 | 4 (Grey) |
| 1 | 0 | 0 | 1 | 1.2 | 0.7 | 4 (White) |
| 2 | 120 | 1 | 1 | 1.35 | 0.7 | 4 (Red) |
| 3 | 240 | 1 | 1 | 1.8 | 0.7 | 4 (Green) |
| 4 | 0 | 1 | 1 | 1.8 | 0.7 | 4 (Blue) |
| 5 | 180 | 1 | 1 | 1.7 | 0.7 | 4 (Yellow) |
| 6 | 0 | 0 | 0.7 | 1.8 | 0.7 | 4 (Grey) |
| 7 | 300 | 1 | 1 | 2.15 | 0.7 | 4 (Cyan) |
| 8 | 60 | 1 | 1 | 1.8 | 0.7 | 4 (Magenta) |
| 9-127 | 0 | 0 | 1 | 1.8 | 0.7 | 4 (White) |

NOTES (continued)

15. Raster Lines Z-value Control: input <15> accepts a real number in the range of 0-1 which is added to the z-values of lines in raster renderings. Sending a 0 to this input will leave lines in their original z position. Sending a 1 to this input will force lines to be in front of everything else in the image. This feature may be desirable when rendering lines exactly along polygon edges. Leaving lines at their original z values will cause obscurity problems with the edges of the the polygons. By adding an offset to the lines' z-values, this obscurity problem is resolved more easily. The default is false.

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB ( %DESCR Name       : P_Varying_Type;
                           Hue        : REAL;
                           Saturation : REAL;
                           Intensity  : REAL;
                           Opaque     : REAL;
                           Diffused   : REAL;          {default .75}
                           Specular   : REAL;          {default 4}
                    Procedure Error_Handler (Err : INTEGER));;
```

## 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 pure blue being 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 intensity. Diffused is the proportion of color contributed by diffuse 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. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

## PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Name := ATTRIBUTES ... AND

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB2 ( %DESCR Name        : P_Varying Type;
                            Hue1        : REAL;
                            Saturation1 : REAL;
                            Intensity1  : REAL;
                            Opaque1     : REAL;
                            Diffused1   : REAL;          {default .75}
                            Specular1   : REAL;          {default 4}
                            Hue2        : REAL;
                            Saturation2 : REAL;
                            Intensity2  : REAL;
                            Opaque2     : REAL;
                            Diffused2   : REAL;          {default .75}
                            Specular2   : REAL;          {default 4}
                   Procedure Error_Handler (Err : INTEGER));;
```

## DEFINITION

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

## PS 300 COMMAND AND SYNTAX

```
Name  := ATTRIBUTES   [COLOR h[,s[i]]]
                      [DIFFUSE d]
                      [SPECULAR s]
                      [OPAQUE t]
         AND          [COLOR h[,s[,i]]]
                      [DIFFUSE d]
                      [SPECULAR s]
                      [OPAQUE t];
```

Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGATR ( CONST Attr : STRING;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

## DEFINITION

This procedure specifies that the attributes named by Attr and specified in a call to PATTRIB or PATTRIB2 apply to all subsequent polygons until superseded by another call to PPLYGATR.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :         :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

## 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 PPLYGATR, PPLYGOTL, AND PPLYGLIS.

Defining a polygon list requires that the user call a minimum of three procedures: PPLYGBEG, to begin the list, one of the list routines (PPLYGLIS, PPLYGRGB, PPLYGHSI) to build the list, and PPLYGEND to end the list. The procedures PPLYGATR and PPLYGOTL provide many options that can be specified when defining a polygon list.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

## Name := POLYGON (END - no corresponding command)

Version A2.V01

### 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 the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :           :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
    PROCEDURE PPlygHsi    (       Coplanar  : Boolean;
                                  NVertices : Integer:
                              VAR Vertices  : P_VectorListType;
                                  NormSpec  : Boolean;
                              VAR Normals   : P_VectorListType;
                                  Colorspec : Boolean
                              VAR Colors    : 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.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- Colors specifies the colors associated with the polygon vertices.
  Colors[n].Draw - Not used
  Colors[n].V4[1] = Hue value mapped to a range 0-360.0;
  Colors[n].V4[2] = Saturation value mapped to range 0-1;
  Colors[n].V4[3] = Intensity value mapped to a range 0-1;

Saturation and intensity values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability rquires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGLIS (       Coplanar  : BOOLEAN;
                           NVertices : INTEGER:
                     CONST Vertices  : P_VectorListType;
                           NormSpec  : Boolean;
                     CONST 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.
  TRUE = specified in the Normals array, FALSE = not specified.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DESCRIPTION

This procedure specifies that Outline to 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 superseded by another call to PPLYGOTL.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
                 :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBList - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGRGB (        Coplanar  : BOOLEAN;
                            NVertices : INTEGER:
                        VAR Vertices  : P_VectorListType;
                            NormSpec  : Boolean;
                        VAR Normals   : P_VectorListType;
                            ColorSpec : Boolean;
                        VAR RGBList   : P_PolyColorType;
                   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.
  TRUE = specified in the Normals array, FALSE = not specified.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- RGBList specifies the colors associated with the polygon vertices.
  RGBList[1,n] = Red
  RGBList[2,n] = Green
  RGBList[3,n] = Blue

  P_PolycolorType is defined as
  P_PolycolorType = ARRAY [1..3, 1..P_MaxpolygonSize] OF INTEGER;

All Red, Green, Blue values are mapped to the range 0-255. Out of range values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability rquires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i];
```

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSUTIL_HSIRGB ( VAR red, green blue:INTEGER;
                          VAR Hue, Saturation, Intensity:REAL);
```

## DEFINITION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

The algorithm used by the PS 340 to covert HSI color specifications to RGB color specifications is adapted from Foley and VanDam's algorithm, which returns RGB values in the range of range 0 to 1 and has the color wheel where Hue=0 is red, Hue=120 is green, Hue=240 is blue. This is NOT the same color wheel specification used by the PS 340.

The PS 340 algorithm returns RGB values as integers in a range of 0 to 255 and has the color wheel where Hue=0 is blue, Hue=120 is red, and Hue=240 is green.

Name := ATTRIBUTES

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

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

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Opaque 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 pure blue being 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 intensity. Diffus is the proportion of color contributed by diffuse 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. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

## PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

# Name := ATTRIBUTES ... AND

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR2 (Name, Hue, Sat, Intens, Opaqul, Diffus, Specul,
             Hue2, Sat2, Inten2, Opaqu2, Diffu2, Specul2, ErrHnd)
```

where

Name is a CHARACTER STRING<br>
Hue is a REAL<br>
Sat is a REAL<br>
Intens is a REAL<br>
Opaqul is a REAL<br>
Diffus is a REAL<br>
Specul is an INTEGER*4<br>
Hue2 is a REAL<br>
Sat2 is a REAL<br>
Inten2 is a REAL<br>
Opaqu2 is a REAL<br>
Diffu2 is a REAL<br>
Specul2 is an INTEGER*4<br>
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 h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t]
        AND        [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

# Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

## 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 superseded by another call to PPLYGA.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i ];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

## 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.

Defining a polygon list requires that the user call a minimum of three subroutines: PPLYGB, to begin the list, one of the list routines (PPLYGL, PPLYGR, PPLYGH) to build the list, and PPLYGE to end the list. The subroutines PPLYGA and PPLYGO provide many options that can be specified when defining a polygon list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGE (ErrHnd)
```

where:

ErrHnd is the user-defineo error-handler subroutine.

## DEFINITION

This subroutine ends the definition of a polygon display list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGH (Coplan, Nverts, Verts, Vedges, NorSpec, Norms,
ColSpe, Colors, 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)
ColSpe is a LOGICAL
Colors 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

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- Colors specifies the colors of the vertices of the polygon. It is of the same form as Verts:
  Colors(1,n) = Hue n
  Colors(2,n) = Saturation n
  Colors(3,n) = Intensity n

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :        :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

## 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

- NVerts specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

(Continued on next page)

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
           :          :           :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

## 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 superseded by another call to PPLYGO.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :         :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBVal - no corresponding command)

Version V2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGR (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ColSpe,
RGBVal, 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)
ColSpe is a LOGICAL
RGBVal is an INTEGER*4(3,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

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

(Continued on next page)

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- RGBVal specified the colors of the vertices of the polygon.
  Colors(1,n) = Red intensity n (range 0..255)
  Colors(2,n) = Green intensity n (range 0..255)
  Colors(3,n) = Blue intensity n (range 0..255)
  Out of range values are converted to the nearest range value without warning.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :         :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSURGB (Red, Green, Blue, Hue, Sat, Intens)
```

where

Red, Green, Blue are INTEGER*4
Hue, Sat, Intens are REAL*4

## DESCRIPTION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce renderings. This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

This algorithm is adapted form Foley and VanDam, which returns RGB values in the range of range 0..1 and has a color wheel where Hue = 0 is red, Hue = 120 is green, Hue = 240 is blue .

This algorithm uses a different color wheel, where Hue = 0 is blue, Hue = 120 is red, and Hue = 240 is green. It returns RGB values as Integers in the range 0..255.

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB ( %DESCR Name       : P_Varying_Type;
                           Hue        : REAL;
                           Saturation : REAL;
                           Intensity  : REAL;
                           Opaque     : REAL;
                           Diffused   : REAL;         {default .75}
                           Specular   : REAL;         {default 4}
                   Procedure Error_Handler (Err : INTEGER));;
```

## 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 pure blue being 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 intensity. Diffused is the proportion of color contributed by diffuse 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. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

## PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Name := ATTRIBUTES ... AND

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB2 ( %DESCR Name        : P_Varying_Type;
                            Hue1        : REAL;
                            Saturation1 : REAL;
                            Intensity1  : REAL;
                            Opaque1     : REAL;
                            Diffused1   : REAL;          {default .75}
                            Specular1   : REAL;          {default 4}
                            Hue2        : REAL;
                            Saturation2 : REAL;
                            Intensity2  : REAL;
                            Opaque2     : REAL;
                            Diffused2   : REAL;          {default .75}
                            Specular2   : REAL;          {default 4}
                    Procedure Error_Handler (Err : INTEGER));;
```

## DEFINITION

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

## PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t]
        AND        [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

## 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 PATTRIB or PATTRIB2 apply to all subsequent polygons until superseded by another call to PPLYGATR.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :           :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

## 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 PPLYGATR, PPLYGOTL, AND PPLYGLIS.

Defining a polygon list requires that the user call a minimum of three routines: PPLYGBEG, to begin the list, one of the list routines (PPLYGLIS, PPLYGRGB, PPLYGHSI) to build the list, and PPLYGEND to end the list. The routines PPLYGATR and PPLYGOTL provide many options that can be specified when defining a polygon list.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

## 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 the procedures required to implement the PS 340 command:

```
Name :=  [WITH [ATTRIBUTES attr] [OUTLINE h]]
         POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
            :          :          :
         [WITH [ATTRIBUTES attr] [OUTLINE h]]
         POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygHsi    (      Coplanar  : Boolean;
                             NVertices : Integer:
                         VAR Vertices  : P_VectorListType;
                             NormSpec  : Boolean;
                         VAR Normals   : P_VectorListType;
                             Colorspec : Boolean
                         VAR Colors    : 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.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- Colors specifies the colors associated with the polygon vertices.
  Colors[n].Draw – Not used
  Colors[n].V4[1] = Hue value mapped to a range 0–360.0;
  Colors[n].V4[2] = Saturation value mapped to range 0–1;
  Colors[n].V4[3] = Intensity value mapped to a range 0–1;

Saturation and intensity values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability rquires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

## 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.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

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

## DESCRIPTION

This procedure specifies that Outline to 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 superseded by another call to PPLYGOTL.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
                  :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBList - no corresponding command)

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGRGB (        Coplanar  : Boolean;
                            NVertices : Integer:
                        VAR Vertices  : P_VectorListType;
                            NormSpec  : Boolean;
                        VAR Normals   : P_VectorListType;
                            ColorSpec : Boolean;
                        VAR RGBList   : P_PolyColorType;
                    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.
  TRUE = specified in the Normals array, FALSE = not specified.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.
- RGBList specifies the colors associated with the polygon vertices.
  RGBList[1,n] = Red
  RGBList[2,n] = Green
  RGBList[3,n] = Blue

P_PolycolorType is defined as
P_PolycolorType = ARRAY [1..3, 1..P_MaxpolygonSize] OF INTEGER;

All Red, Green, Blue values are mapped to the range 0-255. Out of range values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability rquires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

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

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Version A2.V01

## APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSUTIL_HSIRGB ( VAR red, green blue:INTEGER;
                          VAR Hue, Saturation, Intensity:REAL);
```

## DEFINITION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

The algorithm used by the PS 340 to covert HSI color specifications to RGB color specifications is adapted from Foley and VanDam's algorithm, which returns RGB values in the range of range 0 to 1 and has the color wheel where Hue=0 is red, Hue=120 is green, Hue=240 is blue. This is NOT the same color wheel specification used by the PS 340.

The PS 340 algorithm returns RGB values as integers in a range of 0 to 255 and has the color wheel where Hue=0 is blue, Hue=120 is red, and Hue=240 is green.

## Name := ATTRIBUTES

Version A2.V01

### APPLICATION SUBROUTINE AND PARAMETERS

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

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Opaque 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 pure blue being 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 intensity. Diffus is the proportion of color contributed by diffuse 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. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

### PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

## Name := ATTRIBUTES ... AND

Version A2.V01

### APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR2 (Name, Hue, Sat, Intens, Opaqul, Diffus, Specul,
             Hue2, Sat2, Inten2, Opaqu2, Diffu2, Specul2, ErrHnd)
```

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Opaqul is a REAL
Diffus is a REAL
Specul is an INTEGER*4
Hue2 is a REAL
Sat2 is a REAL
Inten2 is a REAL
Opaqu2 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 h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t]
        AND        [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

# Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

## 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 superseded by another call to PPLYGA.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

## 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.

Defining a polygon list requires that the user call a minimum of three subroutines: PPLYGB, to begin the list, one of the list routines (PPLYGL, PPLYGR, PPLYGH) to build the list, and PPLYGE to end the list. The subroutines PPLYGA and PPLYGO provide many options that can be specified when defining a polygon list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

# Name := POLYGON (END - no corresponding command)

Version A2.V01

## 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 the subroutines required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGH (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ColSpe,
Colors, 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)
ColSpe is a LOGICAL
Colors 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

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Vertic.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- Colors specifies the colors of the vertices of the polygon. It is of the same form as Verts:
  Colors(1,n) = Hue n
  Colors(2,n) = Saturation n
  Colors(3,n) = Intensity n

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :        :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

## 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

- NVerts specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name :=  [WITH [ATTRIBUTES attr] [OUTLINE h]]
         POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
            :          :          :
         [[WITH [ATTRIBUTES attr] [OUTLINE r]]
         POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

# Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

## 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 superseded by another call to PPLYGO.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :         :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBVal - no corresponding command)

Version A2.V01

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGR (Coplan, Nverts, Verts, Vedges, NorSpec, Norms,
ColSpe, RGBVal, 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)
ColSpe is a LOGICAL
RGBVal is an INTEGER*4(3,Nverts)
ErrHnd is the user-defined error-handler subroutine.

## DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroucine 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

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

(Continued on next page)

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- RGBVal specifies the colors of the vertices of the polygon.
  Colors(1,n) = Red intensity n (range 0..255)
  Colors(2,n) = Green intensity n (range 0..255)
  Colors(3,n) = Blue intensity n (range 0..255)
  Out of range values are converted to the nearest range value without warning.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

# PSURGB

## APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSURGB (Red, Green, Blue, Hue, Sat, Intens)
```

where

Red, Green, Blue are INTEGER*4
Hue, Sat, Intens are REAL*4

## DESCRIPTION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

The algorithm used by the PS 340 to covert HSI color specifications to RGB color specifications is adapted from Foley and VanDam's algorithm, which returns RGB values in the range of range 0 to 1 and has the color wheel where Hue=0 is red, Hue=120 is green, Hue=240 is blue. This is NOT the same color wheel specification used by the PS 340.

The PS 340 algorithm returns RGB values as integers in a range of 0 to 255 and has the color wheel where Hue=0 is blue, Hue=120 is red, and Hue=240 is green.

# E&S CUSTOMER SERVICE TELEPHONE INFORMATION LIST

Evans & Sutherland Customer Engineering provides a central service number staffed by CE representatives who are available to take requests from 9:00 a.m. Eastern Time to 5:00 p.m. Pacific Time (7:00 a.m. to 6:00 p.m. Mountain Time). All calls concerning customer service should be made to one of the following numbers during these hours. Before you call, please have available your customer site number and system tag number. These numbers are on the label attached to your PS 300 display or control unit.

Customers in the continental United States should call toll-free:

**1 + 800 + 582-4375**

Customers within Utah or outside the continental United States should call:

**(801) 582-5847, Extension 4848**

If problems arise during product installation or you have a question that has not been answered adequately by the customer engineer or the customer service center, contact the regional manager at one of the following Customer Engineering offices:

**Eastern Regional Manager**
**(for Eastern and Central Time Zones)**
**(518) 885-4639**

**Western Regional Manager**
**(for Mountain and Pacific Time Zones)**
**(916) 448-0355**

If the regional office is unable to resolve the problem, you may want to call the appropriate department manager at corporate headquarters:

**National Field Operations**
**(for field service issues)**
**(801) 582-5847, ext 4843**

**Software Support**
**(for sofware issues)**
**(801) 582-5847, ext 4810**

**Technical Support**
**(for hardware issues)**
**(801) 582-5847, ext 4868**

**Director of Customer Engineering**
**(for any unresolved problem)**
**(801) 582-5847, ext 4840**

**READER COMMENT FORM** **Publication Number** ____________________

**Title** ______________________________________________

Your comments will help us provide you with more accurate, complete, and useful documentation. After making your comments in the space below, cut and fold this form as indicated, and tape to secure (please do not staple). This form may be mailed free within the United States. Thank you for your help.

**How did you use this publication?**

- □ General information
- □ Guide to operating instructions
- □ As a reference manual
- □ Other ______________________________

Please rate the quality of this publication in each of the following areas.

| | EXCELLENT | GOOD | FAIR | POOR |
|---|---|---|---|---|
| **Technical Accuracy**<br>Is the manual technically accurate? | □ | □ | □ | □ |
| **Completeness**<br>Does the manual contain enough information? | □ | □ | □ | □ |
| **Readability**<br>Is the manual easy to read and understand? | □ | □ | □ | □ |
| **Clarity**<br>Are the instructions easy to follow? | □ | □ | □ | □ |
| **Organization**<br>Is it easy to find needed information? | □ | □ | □ | □ |
| **Illustrations and Examples**<br>Are they clear and useful? | □ | □ | □ | □ |
| **Physical Attractiveness**<br>What do you think of the overall appearance? | □ | □ | □ | □ |

What errors did you find in the manual? (Please include page numbers) ____________________

**Name** ______________________ **Street** ______________________

**Title** ______________________ **City** ______________________

**Department** ______________________ **State** ______________________

**Company** ______________________ **Zip Code** ____________

All comments and suggestions become the property of Evans & Sutherland.

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 4632 SALT LAKE CITY, UTAH

POSTAGE WILL BE PAID BY ADDRESSEE

**EVANS & SUTHERLAND**
580 Arapeen Drive
Salt Lake City, Utah 84108

ATTN: IAS TECHNICAL PUBLICATIONS

Fold

Cut along dotted line

# USER ERROR REPORTING AND INFORMATIONAL MESSAGES

Supported Under Graphics Firmware A1

# ERROR AND INFORMATIONAL MESSAGES

## CONTENTS

The PS 300 issues four types of messages:

1. Informational messages
2. Warning messages
3. Non-fatal error messages
4. Fatal error messages

Informational messages, warning messages, and non-fatal error messages are all displayed in the same manner on the terminal emulator display. Fatal errors are displayed on PS 300 Keyboard LEDs.

The display of error messages is enabled and disabled by the initial function instance ERROR. The initial function instances WARNING and INFORMATION control the display of warning and informational messages. Refer to the ***PS 300 Function Summary*** for details on these function instances.

To direct one of these types of messages to the host, connect the appropriate output of these function instances to the HOSTOUT initial function instance.

## INFORMATIONAL MESSAGES

*M1000* *REPORT: value =*

This message appears in response to the command:

```
REPORT unit1 AS unit2
```

The user must have previously defined unit1 and unit2 in common terms. For example, if the following command were received by the system:

```
REPORT 1(inch) AS (cm)
```

the following message might be displayed (dependant on actual user-assigned unit definitions):

```
M1000 REPORT: 1(inch) = 2.54(cm)
```

*M1001* *Name ignored*

Issued when a parser error has occurred in a named command, and indicates that nothing is being created under the name. The message also appears when an attempt is made to assign a name to something that cannot be named. For example:

```
A := INIT;
```

would result in:

```
E0001  Parser syntax error = INIT<*>;
M1001  Name ignored: A
```

*M1002* *Depth in BEGIN...END pairs:*

This message appears in response to a COMMAND STATUS request, and reports the depth that has been reached in nested BEGIN...END pairs.

*M1003* *Depth in BEGIN_STRUCTURE...END_STRUCTURE pairs:*

This message appears in response to a COMMAND STATUS request, and reports the depth that has been reached in nested BEGIN_STRUCTURE... END_STRUCTURE pairs.

*M1030* *Strange number of inputs for function named:*

This message appears when one of the functions that allow a user-specified number of inputs or outputs is instanced with more than 127 inputs. If so, the function instance is not created.

For example, this error message would be displayed if the following command were entered:

```
A := F:SYNC(400);
```

*M1086* *File not found*

A syntactically-acceptable file name was specified, but a file was not found for reading.

*M1090* *Hidden-line removal completed*

(PS 340 only) The requested hidden-line rendering has been performed.

*M1091* *Plot finished*

The PS 300 has scaled down the graphics display to fit on the HP plotter, and also scaled down and plotted the terminal emulator contents.

## WARNING MESSAGES

A warning message indicates a condition which is not illegal, but unusual enough to warrant attention. The initial function instance WARNING sends warning messages on output <2>.

*W2021* *Destination of connection does not yet exist: <name>?*

No object was associated with the destination of the connection when the CONNECT command was entered. However, the connection is remembered. For example, the command sequence:

```
A := CHARACTER 'A';
CONNECT A<1>:<1>C;
```

displays the warning message:

```
W2021 Destination of connection does not yet exist: C?
```

but there will be a connection between A and C as soon as an appropriate command for C is entered.

Note that while a connection to a non-existent destination IS made (and results in a warning message), a connection from a non-existent source IS NOT made (and results in an error message: E0075).

*W2023* *Connection already exists*

The specified connection has already been made. For example, the second command in the sequence causes the error message to be displayed:

```
CONNECT A<1>:<1>B;
CONNECT A<1>:<1>B;
```

*W2024* *Msg from <name> discarded --*

No such input. The destination instance exists, but does not have the specified input queue.

No such destination. The destination instance does not exist.

Rejected by destination: <1> <name>? Qtype.

Invalid data has been sent to a SET_CONDITIONAL BIT node or an IF_CONDITIONAL BIT node.

W2041 *Maximum quoted string exceeded; character discarded*

A quoted string in a PS 300 command may contain at most 240 characters. This warning occurs for every character over 240. The string used by the PS 300 will contain only the first 240 characters.

W2042 *Semicolon noticed within quoted string*

This warning generally calls attention to the accidental omission of one of the single quotes delimiting a string. This is a warning and not an error because a semicolon may legally be included within a string.

W2043 *Semicolon noticed within comment*

This warning generally calls attention to the accidental ommission of a closing comment brace. This is a warning and not an error because a semicolon may legally be included within a comment (for example, as in a commented-out command).

W2044 *'{' noticed within comment*

This warning generally calls attention to the accidental omission of a closing comment brace. An opening brace is never legal inside a comment and is discarded.

W2062 *Source of FETCH has no value*

The VARIABLE source for FETCH has not been initialized.

W2070 *Polygon precision error detected; recovery attempted*

(PS 340 only) The hidden-line code has detected a precision error and tried to recover. If the picture is acceptable, this message may be ignored. If the picture has an obvious problem, try a slightly different view.

## NON-FATAL ERRORS

*E0001* *parser syntax error*

Indicates that the syntax of a command or name is not legal. The illegal syntax is displayed at the right of the error message.

For example, if TRI#NGLE were used as a name, a parser syntax error would occur because # is not a legal character for names. In the error message, "<*>" follows the word in which the error was detected.

Parser syntax errors are corrected by reentering proper syntax for the name or command that caused the error.

*E0002* *DEFINE command: undefined unit*

Indicates that an attempt was made to define units in terms of other undefined units. The undefined unit is displayed at the right of the error message.

For example, this message would appear if the following command was received and neither (inch) nor (foot) were defined:

```
DEFINE(FOOT) TO BE 12(INCH);
```

The error is corrected by defining the undefined unit and repeating the DEFINE command.

*E0003* *REPORT command: undefined unit*

Indicates that an attempt was made to report units in terms of units for which no common definition has been established.

The error is corrected by defining both units in terms of a common unit or, more simply, by defining the units in terms of each other, eliminating the need to report.

*E0004* *REPORT command: unit mismatch*

Indicates that an attempt was made to have units reported in terms of units of another measurement type (i.e., converting length to degrees). The mismatched unit is displayed at the right of the error message.

For example, the following command would cause this error message to be displayed:

```
REPORT (FOOT) AS [DEGREES];
```

*E0005* *REPORT command: value = infinity*

Indicates that the result of reporting units in terms of other units is infinity.

For example, this error message would be displayed if the following command were received by the system:

```
REPORT (light years) AS (angstroms);
```

*E0006* *DEFINE command: unit mismatch*

Indicates an attempt was made to define units in terms of units of another measurement type.

For example, this error message would display if the following command were received:

```
DEFINE ()XYZ TO BE 5 [DEGREE];
```

*E0007* *Insufficient privilege*

Indicates the user attempted to use a privileged (restricted) command. In general, privileged commands are for system configuration. (These commands and their syntax are subject to change.)

*E0008* *More END_STRUCTURES than BEGIN_STRUCTURES*

Indicates an attempt to end a grouping without beginning the grouping. The system discards the extra END_STRUCTURE. It is up to the user to check through the command sequences and determine if a BEGIN_STRUCTURE was inadvertently omitted.

*E0009* *More ENDs than BEGINs*

Indicates an attempt to end a grouping without beginning the grouping. The system discards the extra END. It is up to .the user to check through the command sequences and determine if a BEGIN was inadvertently omitted.

*E0010* *Cannot prefix, follow, or remove follower from named element*

Indicates that the applicable element in a PREFIX command, FOLLOW command, or REMOVE FOLLOWER command is data defined by an object modeling command and is therefore illegal.

To correct the error, repeat the command using a legal name or a command other than an object modeling command.

*E0011* *Follower of named element cannot be removed*

Indicates the element named in a REMOVE FOLLOWER command cannot have a follower removed because its follower is an object definition.

*E0012* *Message which function cannot handle*

Indicates that an illegal data type has been sent to a function instance input. The message occurs when the function instance receives the invalid data, not when the connection is made. The name of the function instance and type of the invalid data is displayed to the right of the error message.

The error is corrected by connecting a valid data type to the specified function instance input. The message of illegal type is discarded.

*E0013* *Named item not a display structure*

Indicates that a PREFIX, FOLLOW, REMOVE PREFIX, or REMOVE FOLLOWER command contains the name of something other than a data structure. The invalid name is displayed at the right of the error message.

For example, if SCALE is the name of a function instance and the following command is received:

```
PREFIX SCALE WITH ROTATE IN X 30;
```

The error message reads:

```
E0013  Named item not a display structure SCALE
```

The error is corrected by supplying the name of a display data structure in the command.

*E0014* *Unrecognized function type:*

Indicates an attempt was made to create a function instance from a nonexistent function. The spelling of the nonexistent function is displayed at the right of the error message.

For example, if the following command were received:

```
SPIN := F:TURN;
```

the following error message would be displayed:

```
E0014  Unrecognized function type:  F:TURN
```

*E0016* *ACP timed out; Number of times:*

Indicates the ACP is in a tight loop, usually as a result of trying to process a recursive display data structure.

All user-defined structures are removed from the display when the error is detected. To correct the problem, eliminate the recursive reference and redisplay the structures.

*E0017* *Named item cannot have elements removed or included*

An attempt has been made to INCLUDE data in or REMOVE data from an item other than an object definition.

E0019 *Inconsistent inputs detected by function instance*

Inconsistent data types were sent to a function instance. For example, if an instance of F:ADD receives an integer at input <1> and a vector at input <2>, this message is displayed.

The error message is issued as soon as data which makes the input set inconsistent arrives at an input queue. This can occur before the function has a full firing set.

The message which makes the input set inconsistent is considered to be an invalid message. In accordance with the PS 300's general policy for handling invalid messages, this message is discarded (without restoration of the default value for constant queues). Other messages which existed in the input set before the error message are not considered invalid; however, note that the PS 300 clears ALL of a function's active queues when an invalid message is detected.

E0021 *Error in Binary Data Transfer*

Indicates an error occurred when the user attempted to transfer binary data using the PSIOs or the GSRs.

E0022 *Name not defined or incorrect type: <name>*

The transformed data function (F:XFORM) causes this error to be displayed when a string is sent for which there is no existing node.

E0023 *String overflow; output truncated*

Indicates an attempt to concatenate a string longer than 32,767 characters.

E0029 *Invalid vector index or count*

Indicates that a COPY or SEND to a vector list included an invalid vector index or an invalid count (e.g., a negative integer).

E0032 *Invalid knot sequence*

An invalid knot sequence appears in a BSPLINE or RATIONAL BSPLINE command.

*E0036* *Exhausted working storage*

(PS 340 only) The amount of working storage set aside by the system configurator for surface-rendering operations was not large enough for the attempted operation. Use the RESERVE_WORKING_STORAGE command to increase the amount.

*E0037* *Bad polygon structure or precision error*

(PS 340 only) A bad polygon data definition or precision error was detected during a hidden-line operation.

*E0038* *Sectioning plane not found*

(PS 340 only) A sectioned rendering was requested, but no sectioning plane was found. This can occur if no SECTIONING_PLANE is defined, or if a SECTIONING_PLANE command is an ancestor of a SOLID_RENDERING or SURFACE_RENDERING node or vice versa.

*E0039* *No rendering to save*

(PS 340 only) An attempt at saving a rendering failed because no rendering has yet been requested for the specified marking node.

*E0041* *Type okay but value out-of-range:*

An out-of-range value of the correct data type was received by a function or display data structure.

*E0044* *Hardcopy not initialized*

The hardcopy initialization function was not executed after the plotter was connected.

*E0045* *Allocation error: plotter allocated to another user*

The plotter must be deallocated from its current user and allocated to the user who desires the plot.

*E0046* *Plotter error:*

| | |
|---|---|
| Plotter offline. | The plotter was left offline, possibly when the previous plot was removed. |
| Plotter out of supplies. | The plotter is out of paper or toner. |
| No plotter present. | The plotter has been disconnected, or was never present. |
| Plotter timeout. | Could be due to an improper timeout count or other plotter error. |

*E0047* *Deallocation error: plotter allocated to another user*

The user attempting to deallocate the plotter never had it allocated in the first place.

*E0070* *Generic function not in system; destroying instance*

A valid function name was specified, but that function does not exist in this particular PS 300 configuration.

*E0071* *Source of connection is not a function: <name>?*

Only functions can emit data. So if the object named as the source of a connection is not a function at the time of the CONNECT command, the connection request is rejected. For example, the command sequence:

```
CHARACTER 'A';
CONNECT A<1>:<1>B;
```

would display the error message:

```
E0071 Source of connection is not a function: A?
```

*E0072* *Source of connection is no longer a function: <name>?*

The named function was being deleted at the time the connection was attempted. No connection is made.

*E0073* *No such output source for the function <name>?*

The source of the connection is a function, but it does not have the output specified. For example, the command sequence:

```
A := CHARACTER 'A';
B := F:ADD;
CONNECT B<2>:<1>A;
```

would display the error message since F:ADD has only one output:

```
E0073 No such output source for the function B?<2>
```

*E0074* *Maximum fanout of 127 exceeded -- connection rejected*

A function output may have no more than 127 connections.

*E0075* *Source of connection does not exist: <name>?*

If the named source has not yet been defined, connections from it are ignored. For example, the command sequence

```
A := CHARACTER 'A';
CONNECT C<1>:<1>A;
```

displays the error message:

```
E0075 Source of connection does not exist: C?
```

Note that a connection from a non-existent source is NOT made (and results in an error), while a connection to a non-existent destination IS made (and results in a warning: W2021).

*E0079* *Cannot instantiate with that parameter, so will destroy it*

The "n" parameter value for a function such as F:SYNC(n), F:ROUTE(n), or F:INPUTS_CHOOSE(n) was out-of-range or otherwise invalid. No function instance is created.

*E0080* *String submitted is too long to be a legal name*

The function received a string message containing a node name longer than 254 characters.

*E0085* *Disk directory full, file not written*

Indicates there is insufficient room on the disk for the file.

*E0086* *Disk write error or disk is write protected*

Indicates existance of either a bad disk or no disk in the drive.

*E0087* *Disk read error*

Indicates a bad disk on read.

*E0088* *Bad file name*

A syntactically unacceptable file name was specified.

*E0090* *Can't perform COPY: not a vector list or label*

An attempt has been made to use the COPY command to copy an entity other than an existing vector list or label.

*E0095* *Name must be a function instance*

An attempt has been made to change the queue characteristics of a function whose name is not recognized as a function instance.

*E0097* *Invalid name for save rendering*

Indicates invalid data was sent to save the rendering node.

*E0102* *Cannot affect CNESS for its generic function: <FCNNAME>*

An attempt has been made, using the SETUP CNESS command, to change the queue characteristics of a function which does not allow queue changes.

*E0103* *Corrupt polygon structure*

(PS 340 only) Often indicates a precision error.

*E0104* *Insufficient working storage to remove hidden lines*

(PS 340 only) Use the RESERVE_WORKING_STORAGE command to reserve a larger amount of memory for hidden-line operations.

*E0105* *Cannot complete operation due to insufficient memory*

Indicates that there is insufficient memory to create a data node. The PS 300 terminates the data node.

## FATAL ERRORS

Fatal errors are generally system hardware or software failures from which recovery is not possible. A fatal error may be any one of the following:

- A system error, which is an internal inconsistency detected by PS 300 firmware at a high level (such as a master copy of some function not linked into the system).
- A trap, or an inconsistency, detected by the firmware at a lower level (such as insufficient memory).
- An unexpected hardware exception (as when hardware detects an illegal firmware instruction).

In response to a fatal error:

1. The protocol of each data concentrator is reset by transmitting a null (control-A) to ports 1 through 5, after first lowering the baud rate of any 19200 baud port to 300. This sets up communication between the PS 300 processor and the data concentrator's standard-configuration PS 300 Keyboard.

2. A description of the fatal error is transmitted to ports 1 through 5 and displayed on the function-key LEDs of the PS 300 Keyboard. For system errors and traps, a less detailed error message is also displayed on the PS 300 screen (unless the system hardware has been modified for DEC 56KB communications. Refer to the following section, Warnings and Qualifications).

   The fatal error message consists of a null (control-A), followed by a string which includes the class of fatal error, an error code number, and the location of the firmware's execution when the error was detected. The message is in the form:

   ```
   F9[nnn] *** CRASH. [class] [code] AT [loc]; PLEASE HIT <CTRL P>
   ```

   where:

   [nnn] is 001 for a system error, 002 for a trap, and 003 for an unexpected hardware exception.

   [class] is either SYSTEM ERROR, TRAP, or EXCEPTION.

   [code] is a four-digit hexadecimal number (form $hhhh) for system errors, and a three digit decimal number (form ddd) for traps and exceptions.

   [loc] is an eight-digit hexadecimal number (form $hhhhhhhh).

3. A control-P is solicited ("PLEASE HIT <CTRL P>") on function-key LEDs to determine which port to use for further communications (restarting or debugging). Type <CTRL P> at the port where further communication is to be established.

### NOTE

> If a character other than a <CTRL P> is entered from a port, it is possible that the fatal error handler may misinterpret the baud rate of that port.

4. The user is asked (at the chosen port) if a restart is desired. Answer "Y" to restart (to run initial confidence tests and reload system firmware).

   The PS 300 may also be restarted by turning off system power, waiting a few seconds, then turning system power on. This method of cycling power could aid in clearing a hardware-induced fatal error.

## Warnings and Qualifications

Should a fatal PS 300 error occur, the user should note the failure condition and report it to the E&S Customer Engineering Representative.

### NOTE

> Output of the fatal error message may violate the protocol of the attached device (since the method for resetting the protocol of an arbitrary device cannot be anticipated). For example, the attached device may receive characters even though it had XOFFed the PS 300.

The fatal error handler is oriented towards configurations in which the LEDs of the Evans & Sutherland keyboard are connected to the "A" connector of the data concentrator (standard configuration). A PS 300 Keyboard with LEDs or a terminal (hooked to port 3) is necessary for output display.

Systems having hardware that has been modified for DEC 56Kb interfacing will not display a system error or trap message on the PS 300 screen in the event of a fatal error. (Refer to #2 above.) It is also possible for the debug PROM to be entered prematurely on these systems, depending on whether the DEC 56Kb interface caused/detected the fatal error.

The MEMORY_ALERT function generates a message and a bell alarm when system memory is 75 percent full. The message is updated every 10 seconds. (Both the percentage and the sampling interval for rechecking memory can be changed by the user.)

If the memory falls below the threshhold, the message is removed. If memory depletion is unusually rapid, the system uses all of its resources to recover memory, and does not respond to the user until a sufficient amount of memory is recovered. If memory is truly exhausted, a trap eventually occurs, initiating the four responses to a fatal error described above.

System traps indicate hardware failures that require an E&S Customer Engineer. The one exception may be the following message:

```
No mass memory or too little to initialize
```

If this message appears when initializing the system, the probable cause is a faulty Mass Memory card, requiring a Customer Engineer. If, however, the message appears when the system has been running, the probable cause is an overloaded Mass Memory that resulted from trying either to load too much data or to improperly connect function networks. Check all function networks for proper connections.

If any message appears other than described in preceding sections or in the above trap message, note the message and report it to your E&S Customer Engineer.