DORÉ

DORÉ REFERENCE MANUAL

Stardent Computer Inc.

Part Number: 340-0108-01

Change History

340-0004-02	Original
340-0004-03	February, 1989 – Software Release 2.0
340-0108-00	December, 1989 – Software Release 3.0

CONTENTS

PREFACE

This is a reference manual for Doré, a three-dimensional computer graphics subroutine library. For detailed examples of Doré application programs, refer to the *Doré Programmer's Guide*. This book is a reference for programmers already familiar with Doré fundamentals.

This manual has the following chapters:

- Chapter 1, *Doré Functions*, presents the Doré functions in "man" page format. The functions are listed alphabetically by their C function call names.
- Chapter 2, *C Include Files*, lists the data type structures, constants, and function declarations used by Doré applications written in C. Function prototypes for Doré are also included in this chapter.
- Chapter 3, *Fortran Include Files*, lists the data types, constants, and function declarations used by Doré applications written in Fortran.
- Chapter 4, *Quick Reference*, lists each Doré function with its arguments and a one-line description.
- Chapter 5, *Matrix Transformations*, explains how Doré manipulates its transformation matrix and shows the matrix equations that are used by Doré functions.
- Appendices A and B, *Doré Naming Conventions* and *Doré Vertex Types*, list some general information about Doré function names and primitive object vertex considerations.
- Appendix C, Raster File and Memory Formats, describes the file and memory data format for raster data used by Doré.

- Appendices D, E, and F, *Doré Renderers, Doré Configurations* and *Doré Device Drivers*, contain detailed information about using Doré in particular configurations. Add any extra vendor specific documentation that pertains to your Doré system to these appendices.

Intended Audience

This manual is written for the experienced programmer who is already comfortable writing graphics applications. The man pages provide the calling sequences and applicable return values when the Doré Library is called from the C and Fortran programming languages. If you are unfamiliar with C or Fortran, or with computer graphics programming, consult the related readings listed below.

Related Readings

Programming guides for experienced programmers who have never programmed graphics applications:

- *Fundamentals of Interactive Computer Graphics* by Foley and Van Dam, published by Addison Wesley
- *Principles of Interactive Computer Graphics* by Newman and Sproull, published by McGraw Hill
- *Procedural Elements for Computer Graphics* by Rogers, published by McGraw Hill

Programming guide for the Doré Library:

- *Doré Programmer's Guide* by Stardent Computer Inc. Computer Corporation

Programming reference for the C programming language:

- *C: A Reference Manual* by Harbison and Steele, published by Prentice Hall

Programming reference for the Fortran programming language:

- *Fortran Reference Manual* by Stardent Computer Inc. Computer Corporation
- *A Guidebook to Fortran on Supercomputers* by John M. Levesque and Joel W. Williamson, published by Academic

DORÉ FUNCTIONS

CHAPTER ONE

The following pages present the Doré Library functions in "man" page format. They are ordered alphabetically by the C function names. Both the C and the Fortran function syntaxes are listed at the top of each page. The Fortran constant names are given in angle brackets directly after their C equivalents. For example:

The default value for *DoRepType* is *DcWireframe* <*DCWIRE*>.

Some of the Doré Library functions have a different calling sequence when used in Fortran application programs. These functions' manual pages contain extra information detailing the proper Fortran calling syntax.

NAME

DdInqColorEntries – Return color lookup table entries of a device

SYNOPSIS

C:

```
void DdInqColorEntries(device, colormodel, start, count, entries)
DtObject device;
DtColorModel colormodel;
DtInt start;
DtInt count;
DtReal entries[];
```

Fortran:

```
CALL DDQCE(DEVICE, COLMOD, START, COUNT, ENTRYS)
INTEGER*4 DEVICE
INTEGER*4 COLMOD
INTEGER*4 START
INTEGER*4 COUNT
REAL*8 ENTRYS(*)
```

DESCRIPTION

DdInqColorEntries returns in the supplied array *entries* all the color entries in the color lookup table starting at the *start* location for *count* number of entries on the specified device *device*. Each entry's length is determined by the *colormodel* parameter, which specifies the color model used to represent color information.

Many actual devices use a color lookup table or a fixed set of colors or grey values. *DdInqColorEntries* allows the user to determine the values of the user-writable or fixed set of tables.

ERRORS

DdInqColorEntries will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdInqColorEntries will also fail if the *start* or *count* parameters refer to entries outside the color lookup table boundaries.

[WARNING - bad start and/or count values]

SEE ALSO

DdSetColorEntries(3D), DdInqColorTableSize(3D), DdInqVisualType(3D)

NAME

DdInqColorTableSize – Return the number of entries in the color lookup table

SYNOPSIS

C:

```
DtInt DdInqColorTableSize(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQCTS(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqColorTableSize returns the number of entries in the color lookup table for the device, *device*. The color table defines the mapping between pixel values and colors displayed on the screen.

DdInqColorEntries <*DDQCE*> may be used to examine the values in the color lookup table. *DdSetColorEntries* <*DDSCE*> may be used to set the entries in the lookup table (on devices that have writable tables: *DcGrayScale* <*DCGRYS*>, *DcPseudoColor* <*DCPSUC*>, and *DcTrueColor* <*DCTRUC*>).

ERRORS

DdInqColorTableSize will fail if the device handle is invalid; the value -1 is returned.

[WARNING - invalid device handle]

SEE ALSO

DdInqColorEntries(3D), DdSetColorEntries(3D)

NAME

DdInqExtent – Return the extent of a device

SYNOPSIS

C:

```
void DdInqExtent(device, volume)
DtObject device;
DtVolume *volume;
```

Fortran:

```
CALL DDQE(DEVICE, VOLUME)
INTEGER*4 DEVICE
REAL*8 VOLUME(6)
```

DESCRIPTION

DdInqExtent gets the volume range available to an indicated device, *device*. It returns a value in the *volume* parameter supplied.

ERRORS

DdInqExtent will fail if the device handle is invalid.

[WARNING - invalid device handle]

SEE ALSO

DdInqViewport(3D), DdInqResolution(3D), DdInqVisualType(3D), DoDevice(3D)

NAME

DdInqFonts – Return the set of fonts supported by a device

SYNOPSIS

C:

```
void DdInqFonts(device, fonts)
DtObject device;
DtFontPrecision fonts[ ];
```

Fortran:

```
CALL DDQFT(DEVICE, FONTS)
INTEGER*4 DEVICE
INTEGER*4 FONTS(*)
```

DESCRIPTION

DdInqFonts returns the set of fonts supported on a device, *device*. The array *fonts* consists of a set of pairs. Each pair specifies a fontid and a corresponding precision. The possible values for the text precision are: *DcStringPrecision* <*DCSTRP*>, *DcCharacterPrecision* <*DCCHRP*>, and *DcStrokePrecision* <*DCSTKP*>. Stroke precision is considered the most complex, and string precision is considered the least complex. If a font is available at a specified precision, it is also available at all lower precisions.

ERRORS

DdInqFonts will fail if the device handle is invalid.

[WARNING - invalid device handle]

SEE ALSO

DdInqNumFonts(3D)

NAME

DdInqFrame – Return the frame of a device

SYNOPSIS

C:

```
DtObject DdInqFrame(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQFR(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqFrame returns the frame for device, *device*. Only one frame is allowed per device.

A frame is an organizational object used to describe an image that is to be displayed on a device. A frame can be displayed on zero or more devices using the function *DdSetFrame <DDSF>*.

ERRORS

DdInqFrame will fail if the device handle is invalid; the value *DcNullObject <DCNULL>* is returned.

[WARNING - invalid device handle]

SEE ALSO

DdSetFrame(3D)

NAME

DdInqNumFonts – Return the number of fonts supported by a device

SYNOPSIS

C:

```
DtInt DdInqNumFonts(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQNF(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqNumFonts returns the number of fonts supported by a device, *device*.

ERRORS

DdInqNumFonts will fail if the device handle is invalid; the value -1 is returned.

[WARNING - invalid device handle]

SEE ALSO

DdInqFonts(3D)

NAME

DdInqPickAperture – Return the pick aperture of a device

SYNOPSIS

C:

```
void DdInqPickAperture(device, aperture)
DtObject device;
DtSize3 *aperture;
```

Fortran:

```
CALL DDQPA(DEVICE, APRTUR)
INTEGER*4 DEVICE
REAL*8 APRTUR(3)
```

DESCRIPTION

DdInqPickAperture returns the pick aperture of a specified device, *device*, into the *width*, *height*, and *depth* fields of the supplied argument *aperture* <in Fortran, the aperture is returned in an array>.

When a pick is initiated on a device (via *DdPickObjs* <*DDPO*>), a volume of space centered around the pick point is searched for drawable objects. The dimensions of that volume are defined by the device's pick aperture as specified by a width, a height, and a depth in device coordinates. The pick aperture is set by *DdSetPickAperture* <*DDSPA*>.

ERRORS

DdInqPickAperture will fail if passed an invalid device handle.

[WARNING - invalid device handle]

SEE ALSO

DdPickObjs(3D), DdSetPickAperture(3D)

NAME

DdInqPickCallback – Return the pick callback object of a device

SYNOPSIS

C:

```
DtObject DdInqPickCallback(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQPC(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqPickCallback returns the pick callback object used on the device, *device*.

Picking is a method of identifying displayable primitives that fall within a specified volume known as the device pick aperture. A pick is initiated by a call to *DdPickObjs <DDPO>*. The pick callback is a callback object that selects which primitives found within the pick aperture during a pick (known as "hits") will actually be returned. Doré currently provides three standard pick callback objects that may be used; however users may also easily provide their own. The standard call back objects are:

DcPickFirst <DCPKFR>
This callback causes *DdPickObjs* to accept only the first hit that it finds.

DcPickClosest <DCPKCL>
This callback causes *DdPickObjs* to accept only the hit that was frontmost (closest to the viewer) in the pick aperture.

DcPickAll <DCPKAL>
This callback causes *DdPickObjs <DDPO>* to add all hits found to the hit list.

Users may include functions that accept or exclude only hits on certain types of objects or ones that only accept up to a maximum number of hits before terminating a search. See *DdSetPickCallback* for information on creating pick callbacks.

ERRORS

DdInqPickCallback will fail if passed an invalid device handle; the value *DcNullObject <DCNULL>* is returned.

[WARNING - invalid device handle]

SEE ALSO

DdSetPickCallback(3D), DdPickObjs(3D), DdSetPickAperture(3D), DdInqPickAperture(3D), DdSetPickPathOrder(3D), DdInqPickPathOrder(3D), DoPickId(3D)

NAME

DdInqPickPathOrder – Return the pick path order of a device

SYNOPSIS

C:

```
DtPickPathOrder DdInqPickPathOrder(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQPPO(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqPickPathOrder returns the pick path order on the device, *device*. A device's pick path order specifies the order in which elements in the pick paths are returned on a device, *device*, by *DdPickObjs* <*DDPO*>. See *DdSetPickCallback* for a description of pick paths.

The two possible values for a device's pick path order are *DcTopFirst* <*DCTOPF*> and *DcBottomFirst* <*DCBOTF*>. *DcTopFirst* <*DCTOPF*> means that the pick path starts with the largest group within the view containing the picked primitive object and ends with the picked primitive object itself. *DcBottomFirst* <*DCBOTF*> is just the opposite.

Note: the value of the pick path order affects only the order of pick path elements in paths returned from *DdPickObjs* <*DDPO*> and does NOT affect the order of elements as seen by the pick callback functions (see *DdSetPickCallback*).

ERRORS

DdInqPickPathOrder will fail if passed an invalid device handle; the returned value is undefined.

[WARNING - invalid device handle]

SEE ALSO

DdSetPickPathOrder(3D), DdPickObjs(3D), DdSetPickCallback(3D)

NAME

DdInqPixelData – Return pixel information about an image on a given device

SYNOPSIS

C:

```
DtFlag DdInqPixelData(device, requesttype, width, height, type, data,
                      userdelete)
DtObject device;
DtRasterType requesttype;
DtInt *width;
DtInt *height;
DtRasterType *type;
DtPtr *data;
DtFlag *userdelete;
```

Fortran:

```
INTEGER*4 DDQPXD(DEVICE, REQTYP, WIDTH, HEIGHT, TYPE,
                 DATA, USRDEL)
INTEGER*4 DEVICE
INTEGER*4 REQTYP
INTEGER*4 WIDTH
INTEGER*4 HEIGHT
INTEGER*4 TYPE
INTEGER*4 DATA
INTEGER*4 USRDEL
```

DESCRIPTION

DdInqPixelData returns pixel information about an image on the device *device*. This infomation can be used to create a raster object with *DoRaster <DORS>*. *DdInqPixelData* will return *DcFalse <DCFALS>* if the device cannot return the pixel information.

The return parameters *width* and *height* are the dimensions of the image.

The parameter *requesttype* is the type and format of pixel information requested. The return parameter *type* is the type and format of the pixel information returned. Possible values for *requesttype* and *type* are:

DcRasterRGB <DCRRGB>
Each pixel has red, green and blue information.

DcRasterRGBA <DCRRA>
Each pixel has red, green, blue and alpha information.

DcRasterRGBAZ <DCRRAZ>
Each pixel has red, green, blue, alpha and z information.

DcRasterRGBZ <DCRRZ>
Each pixel has red, green, blue and z information.

DcRasterA <DCRA>
Each pixel has alpha information.

DcRasterZ <DCRZ>
Each pixel has z information.

The return parameter *data* is a pointer to the pixel data. The return parameter *userdelete* is a flag specifying whether the application is responsible for deallocating the space pointed to by *data*. Possible values for *userdelete* are:

DcTrue <*DCTRUE*>
The application is responsible for deleting the space pointed to by *data* when it no longer needs the data.

DcFalse <*DCFALS*>
The application must not delete the space pointed to by *data*.

ERRORS

DdInqPixelData will fail if passed an invalid device handle.

[WARNING - invalid device handle]

SEE ALSO

DoRaster(3D)

NAME

DdInqResolution – Return the resolution of a device

SYNOPSIS

C:

```
void DdInqResolution(device, x, y)
DtObject device;
DtReal *x;
DtReal *y;
```

Fortran:

```
CALL DDQR(DEVICE, X, Y)
INTEGER*4 DEVICE
REAL*8 X
REAL*8 Y
```

DESCRIPTION

DdInqResolution returns the size of a pixel (in millimeters) on the device, *device*. It returns the width and height values in the two parameters *x* and *y* supplied.

DdInqExtent <*DDQE*> is used to determine the extent of the device in XYZ.

ERRORS

DdInqResolution will fail if the device handle is invalid.

[WARNING - invalid device handle]

SEE ALSO

DdInqExtent(3D)

NAME

DdInqShadeMode – Return the shade mode of a device

SYNOPSIS

C:

```
DtShadeMode DdInqShadeMode(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQSM(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqShadeMode returns the shade mode of a device, *device*.

Doré can produce output on devices that have limited color capability. These devices are usually referred to as "pseudocolor" devices. Doré always computes shades in full color, and compresses these shades to pseudocolor for output.

The shade mode specifies how this compression is performed. This attribute can be either *DcComponent* <*DCCOMP*> or *DcRange* <*DCRNG*>. A value of *DcComponent* <*DCCOMP*> means that each primary computed color (red, green, and blue) is compressed into a limited bit range and the resulting bit fields are Or'ed together to form a single n-bit value. The compression algorithm is fixed for a specific value of n (for n = 8; 3 bits red, 3 bits green, 2 bits blue).

A value of *DcRange* <*DCRNG*> specifies that the user has loaded a particular color map containing known ranges of shading (like gold and silver) into the pseudocolor device. The *DdInqShadeRanges* <*DDQSR*> and *DdSetShadeRanges* <*DDSSR*> calls inquire and setup the contents of specific shade ranges. See these manual pages for further information on this pseudocolor mode.

ERRORS

DdInqShadeMode will fail if the device handle is invalid; the returned value is undefined.

[WARNING - invalid device handle]

SEE ALSO

DdInqColorEntries(3D), DdInqColorTableSize(3D), DdInqShadeRanges(3D), DdInqVisualType(3D), DdSetColorEntries(3D), DdSetShadeMode(3D), DdSetShadeRanges(3D), DoShadeIndex(3D), DoDevice(3D)

DdInqShadeRanges(3D)

NAME

DdInqShadeRanges – Return the shade range table entries of a device

SYNOPSIS

C:

```
void DdInqShadeRanges(device, start, count, entries)
DtObject device;
DtInt start;
DtInt count;
DtInt entries[];
```

Fortran:

```
CALL DDQSR(DEVICE, START, COUNT, ENTRYS)
INTEGER*4 DEVICE
INTEGER*4 START
INTEGER*4 COUNT
INTEGER*4 ENTRYS(COUNT*2)
```

DESCRIPTION

DdInqShadeRanges is used to query one or more shade range table entries from a device, *device*.

Each entry in the shade range table consists of a pair of values defining a shade range. These values represent the color table index of the beginning and end of the range. One value represents the minimum shade value in a range, and the other represents the maximum shade value in a range. Intensities between 0.0 and 1.0 are used to linearly interpolate between these two specified limits in the color lookup table. This interpolated color table index is then sent to the device's frame buffer.

For a given shade range *i*, *entries[2*i]* is the start, or minimum value of the range, and *entries[2*i+1]* is the end, or maximum value of the range.

The parameter *start* specifies the first entry to be queried. *count* specifies the number of entries to be read. *entries* is an array containing the queried entries.

ERRORS

DdInqShadeRanges will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdInqShadeRanges will also fail if the *start* or *count* parameters refer to entries outside the shade range table boundries.

[WARNING - bad start and/or count values]

SEE ALSO

DdInqColorEntries(3D), DdInqColorTableSize(3D), DdInqShadeMode(3D), DdInqVisualType(3D), DdSetColorEntries(3D), DdSetShadeMode(3D), DdSetShadeRanges(3D), DoShadeIndex(3D)

NAME

DdInqViewport – Return the size of the device viewport of a device

SYNOPSIS

C:

```
void DdInqViewport(device, viewport)
DtObject device;
DtVolume *viewport;
```

Fortran:

```
CALL DDQV(DEVICE, VWPORT)
INTEGER*4 DEVICE
REAL*8 VWPORT(6)
```

DESCRIPTION

DdInqViewport returns in the supplied parameter *viewport* the extent of the device viewport for the device, *device*.

The device viewport is the subportion of the device's extent (the display volume is obtainable with *DdInqExtent* <*DDQE*>) upon which the application displays the device's frame.

The default viewport uses all the current device volume. If the device extent changes (for example, if an X-window is resized), then by default, the viewport automatically changes to fit the new device extent. The auto-resize feature can be disabled for a device.

ERRORS

DdInqViewport will fail if the device handle is invalid.

[WARNING - invalid device handle]

SEE ALSO

DdSetViewport(3D), DdInqExtent(3D), DoDevice(3D)

NAME

DdInqVisualType – Return the visual type of a device

SYNOPSIS

C:

```
DtVisualType DdInqVisualType(device)
DtObject device;
```

Fortran:

```
INTEGER*4 DDQVT(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdInqVisualType returns the visual type of the specified device, *device.* (See *DoDevice*).

Doré defines the following visual types:

DcPseudoColor <DCPSUC>
A pixel value indexes a color map to determine the color to be displayed; the entries in the color map can be changed dynamically.

DcGreyScale <DCGRYS>
These devices are the same as pseudocolor devices, except that only the intensities of the entries specified in the lookup table are displayed.

DcDirectColor <DCDRCC>
A pixel value is decomposed into separate subfields, and each subfield separately indexes the color map for the corresponding value. The entries in the color map can be changed dynamically. Typically, the color map is used for displaying false color or for performing gamma correction.

DcTrueColor <DCTRUC>
A true color device is similar to a directcolor device except that the color map has predefined read-only values that provide a (near-) linear ramp in each color coordinate (e.g. a direct RGB device).

DcStaticColor <DCSTCC>
A static color device is similar to a pseudocolor device except that the color map has predefined read-only values that are device dependent (such as a fixed 16 color system).

DcStaticGrey <DCSTCG>
A static grey scale device is similar to a static color device except that only a fixed set of device-dependent intensities is supported. (A "black and white only" display is of this type.)

ERRORS

DdInqVisualType will fail if the device handle is invalid; the returned value is undefined.

[WARNING - invalid device handle]

SEE ALSO

DdInqColorEntries(3D), DdInqColorTableSize(3D), DoDevice(3D)

NAME

DdPick – This function has been replaced by *DdPickObjs*

SYNOPSIS

C:

```
void DdPick(device, pick_point, hit_count, index_size, index,
        list_size, hit_list, z_values, views, error_word)
DtObject device;
DtPoint3 pick_point;
DtInt *hit_count;
DtInt index_size;
DtInt index[];
DtInt list_size;
DtInt hit_list[];
DtReal z_values[];
DtObject views[];
DtInt *error_word;
```

Fortran:

```
CALL DDP(DEVICE, PICKPT, HITCNT, IDXSIZ, INDEX,
        LSTSIZ, HITLST, ZVALUS, VIEWS, ERRWRD)
INTEGER*4 DEVICE
REAL*8 PICKPT(3)
INTEGER*4 HITCNT
INTEGER*4 IDXSIZ
INTEGER*4 INDEX(*)
INTEGER*4 LSTSIZ
INTEGER*4 HITLST(*)
REAL*8 ZVALUS(*)
INTEGER*4 VIEWS(*)
INTEGER*4 ERRWRD
```


DESCRIPTION

DdPick initiates a pick on a device. Picking is a method of identifying the drawable objects that fall within a specified volume known as a pick aperture (see *DdSetPickAperture*). These objects are known as "hits" and are uniquely identified by a pick path (see *DdSetPickCallback* for the definition of a pick path). Picking is usually initiated by a user, via an input device, who wants to address a particular object on the screen.

pick_point
: A coordinate triple that specifies a point in device coordinates around which to search for hits.

hit_count
: The number of hits found within the pick aperture whose paths are being returned.

index_size
: The size of the array *index*.

index
: A pointer to user supplied space large enough for *index_size* number of 32-bit integers.

list_size
: The size of the array *hit_list*.

hit_list

A pointer to user supplied space (an array) large enough for *list_size* number of pick path elements. Note that a pick path element is three *DtInt*s large. Returned in the array *index* are the indexes into the *hit_list* array of the beginnings of the pick paths being returned. The last element in the path beginning at *hit_list*[*index*[i]] is always one element before *hit_list*[*index*[i+1]]. This is true for the last path as well, which means that if the application expects to find "N" hits, it must provide an index array with a size of at least "N + 1."

z_values

An optional pointer to user supplied space (array) large enough for (*index_size* - 1) *DtReal*s. Returned here are the depth values in picking coordinates units of the closest point within the pick aperture of the corresponding hits whose paths are referred to in *index*. The value *DcNullPtr* <*DCNULL*> may be given if the user is not concerned with the depth values.

views

An optional pointer to user supplied space (array) large enough for (*index_size* - 1) *DtObject*ss. Returned here is an array of view objects that corresponds to the views in which the hits were found. Again, the value *DcNullPtr* <*DCNULL*> may be given if the user is not concerned with views.

error_word

The returned error word.

The value of the error word returned is a bit field that is the logical OR of zero or more of the following constants:

DcPickBadStatus <*DCPBAD*>
DcPickListOverflow <*DCPLOV*>
DcPickIndexOverflow <*DCPIOV*>

A bad status error results from an illegal *pick control status* returned by a pick callback (see *DdSetPickCallback*). List and index overflows occur when either the array *hit_list* or *index* are too small to accommodate another hit. In all these cases, the user is still returned data from the previous hits that were recorded. Application programs may still wish to use this data or, if the errors result from lack of space, they may decide either to allocate more space or to use a smaller pick aperture and then try the same pick again.

ERRORS

DdPick will fail if passed an invalid device handle.

[WARNING - invalid device handle]

DdPick will fail if *index_size* or *list_size* are invalid

[WARNING - non-positive index or list size]

DdPick will fail if Doré is already performing a traversal of the database

[WARNING - traversal already in progress]

SEE ALSO

DdInqPickAperture(3D), DdInqPickCallback(3D), DdInqPickPathOrder(3D), DdSetPickAperture(3D), DdSetPickCallback(3D), DdSetPickPathOrder(3D), DoFilter(3D), DoPickID(3D), DoPickSwitch(3D)

NAME

DdPickObjs – Initiate a pick on a device

SYNOPSIS

C:

```
void DdPickObjs(device, pick_point, hit_count, index_size, index,
        list_size, hit_list, z_values, wcs_values,
            lcs_values, views, error_word)
DtObject device;
DtPoint3 pick_point;
DtInt *hit_count;
DtInt index_size;
DtInt index[];
DtInt list_size;
DtInt hit_list[];
DtReal z_values[];
DtReal wcs_values[];
DtReal lcs_values[];
DtObject views[];
DtInt *error_word;
```

Fortran:

```
CALL DDPO(DEVICE, PICKPT, HITCNT, IDXSIZ, INDEX,
        LSTSIZ, HITLST, ZVALUS, WCSVAL,
            LCSVAL, VIEWS, ERRWRD)
INTEGER*4 DEVICE
REAL*8(3) PICKPT
INTEGER*4 HITCNT
INTEGER*4 IDXSIZ
INTEGER*4 INDEX(*)
INTEGER*4 LSTSIZ
INTEGER*4 HITLST(*)
REAL*8 ZVALUS(*)
REAL*8 WCSVAL(*)
REAL*8 LCSVAL(*)
INTEGER*4 VIEWS(*)
INTEGER*4 ERRWRD
```

DESCRIPTION

DdPickObjs initiates a pick on a device. Picking is a method of identifying the drawable objects that fall within a specified volume known as a pick aperture (see *DdSetPickAperture*). These objects are known as "hits" and are uniquely identified by a pick path (see *DdSetPickCallback* for the definition of a pick path). Picking is usually initiated by a user, via an input device, who wants to address a particular object on the screen.

pick_point
A coordinate triple that specifies a point in device coordinates around which to search for hits.

hit_count
The integer count of the number of hits found within the pick aperture whose paths are being returned.

index_size
The size of the array *index*.

index
A pointer to user supplied space large enough for *index_size* number of 32-bit integers.

list_size
The size of the array *hit_list*.

hit_list
A pointer to user supplied space (an array) large enough for *list_size* number of pick path elements. Note that a pick path element is three *DtInts* large. Returned in the array *index* are the indexes into the *hit_list* array of the beginnings of the pick paths being returned. The last element in the path beginning at *hit_list*[*index*[i]] is always one element before *hit_list*[*index*[i+1]]. This is true for the last path as well, which means that if the application expects to find "N" hits, it must provide an index array with a size of at least "N + 1".

z_values
An optional pointer to user supplied space (array) large enough for *index_size* - 1 *DtReal*'s. Returned here are the depth values in device coordinate units of the closest point within the pick aperture of the corresponding hits whose paths are referred to in *index*. The value *DcNullPtr* <*DCNULL*> may be given if the user is not concerned with the depth values.

wcs_values
An optional pointer to user supplied space (array) large enough for 3*(*index_size* - 1) *DtReal*'s. Returned here are the world coordinate values of the closest point within the pick aperture of the corresponding hits whose paths are referred to in *index*. The world coordinate of hit *n* is *wcs_values[n*3], wcs_values[n*3+1], wcs_values[n*3+2]*. The value *DcNullPtr* <*DCNULL*> may be given if the user is not concerned with the world coordinate values.

lcs_values
An optional pointer to user supplied space (array) large enough for 3*(*index_size* - 1) *DtReal*'s. Returned here are the local coordinate values of the closest point within the pick aperture of the corresponding hits whose paths are referred to in *index*. The local coordinates are the coordinates in which the object's geometry was defined. The local coordinate of hit *n* is *lcs_values[n*3], lcs_values[n*3+1], lcs_values[n*3+2]*. The value *DcNullPtr* <*DCNULL*> may be given if the user is not concerned with the local coordinate values.

views
An optional pointer to user supplied space (array) large enough for *index_size* - 1 *DtObjects*'s. Returned here is an array of view objects that corresponds to the views in which the hits were found. Again, the value *DcNullPtr* <*DCNULL*> may be given if the user is not concerned with views.

error_word
The address at which an error word is returned.

The value of the error word returned is a bit field that is the logical "or" of zero or more of the following constants:

DcPickBadStatus <DCPBAD>
A bad status error results from an illegal *pick control status* returned by a pick callback (see *DdSetPickCallback*).

DcPickListOverflow <DCPLOV>
A list overflow occurs when the array *hit_list* is too small to accommodate another hit.

DcPickIndexOverflow <DCPIOV>
An index overflow occurs when the array *index* is too small to accommodate another hit.

In all these cases, the user is still returned data from the previous hits that were recorded. Application programs may still wish to use this data or, if the errors result from lack of space, they may decide either to allocate more space or to use a smaller pick aperture and then try the same pick again.

ERRORS

DdPickObjs will fail if passed an invalid device handle.

[WARNING - invalid device handle]

SEE ALSO

DdInqPickAperture(3D), DdInqPickCallback(3D), DdInqPickPathOrder(3D), DdSetPickAperture(3D), DdSetPickCallback(3D), DdSetPickPathOrder(3D), DoFilter(3D), DoPickID(3D), DoPickSwitch(3D)

NAME

DdSetColorEntries – Set the color lookup table entries of a device

SYNOPSIS

C:

```
void DdSetColorEntries(device, colormodel, start, count, entries)
DtObject device;
DtColorModel colormodel;
DtInt start;
DtInt count;
DtReal entries[];
```

Fortran:

```
CALL DDSCE(DEVICE, COLMOD, START, COUNT, ENTRYS)
INTEGER*4 DEVICE
INTEGER*4 COLMOD
INTEGER*4 START
INTEGER*4 COUNT
REAL*8 ENTRYS(*)
```

DESCRIPTION

DdSetColorEntries sets the color entries in the color lookup table of a pseudocolor device, *device*, starting at the *start* location for *count* number of entries in the array, *entries*. Many actual devices use a color lookup table or a fixed set of colors or grey values. *DdSetColorEntries* allows the user to specify a set of colors for these tables if they are user-writable.

colormodel
Specifies the color model of the colors in the array *entries*.

start
Specifies the beginning entry.

count
Specifies the total number of colors to be set.

entries
The array of values for the kind of color specified by *colormodel*. For example, *entries* of color values for *colormodel DcRGB <DCRGB>* would take three components: a red value, a green value, and a blue value.

ERRORS

DdSetColorEntries will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdSetColorEntries will fail if the *start* or *count* parameters refer to entries outside the color lookup table boundries.

[WARNING - bad start and/or count values]

SEE ALSO

DdInqColorEntries(3D), DdInqColorTableSize(3D)

NAME

DdSetFrame – Attach a frame to a device

SYNOPSIS

C:

```
void DdSetFrame(device, frame)
DtObject device;
DtObject frame;
```

Fortran:

```
CALL DDSF(DEVICE, FRAME)
INTEGER*4 DEVICE
INTEGER*4 FRAME
```

DESCRIPTION

DdSetFrame sets the frame, *frame*, for a device, *device*. Only one frame at a time is allowed per device. A frame is an organizational object used to describe an image composed from one or more views. A frame can be displayed on one or more devices.

If a frame is already attached to a particular device when this routine is called, the old frame is detached and the new frame is attached in its place. To detach a frame from a given device, pass a *DcNullObject* <*DCNULL*> for the frame parameter.

The frame defines a device independent 3-D local coordinate system within which one or more views can be placed. By default the frame extends from (0.0, 0.0, 0.0) to (1.0, 1.0, 1.0). The frame boundary can be queried using *DfInqBoundary* <*DFQB*>.

When the frame is displayed on a device the frame boundary is mapped onto the largest right rectangular volume that can fit within the device viewport such that aspect ratio in X and Y is preserved and the Z extent of the frame is mapped to the entire Z extent of the device. When this mapping is performed, there may be extra "white space" inside the device viewport. The function *DfSetJust* <*DFSJ*> can be used to allocate this white space.

ERRORS

DdSetFrame will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdSetFrame will fail if the frame handle is invalid.

[WARNING - invalid frame handle]

SEE ALSO

DdInqFrame(3D), DoFrame(3D)

NAME

DdSetPickAperture – Set the pick aperture of a device

SYNOPSIS

C:

```
void DdSetPickAperture(device, aperture)
DtObject device;
DtSize3 *aperture;
```

Fortran:

```
CALL DDSPA(DEVICE, APRTUR)
INTEGER*4 DEVICE
REAL*8 APRTUR(3)
```

DESCRIPTION

DdSetPickAperture sets the pick aperture on a specified device, *device*, for use during picking.

When a pick is initiated on a device (via *DdPickObjs <DDPO>*), a volume of space centered around the pick point is searched for drawable objects. The dimensions of that volume define that device's pick aperture which is always at right angles to the device. The *width*, *height*, and *depth* fields <three consecutive values in a Fortran array> are given in device coordinates where one unit is the distance between adjacent pixel centers.

ERRORS

DdSetPickAperture will fail if passed an invalid device handle.

[WARNING - invalid device handle]

DEFAULTS

The default pick aperture is 10.0 (X) by 8.0 (Y) by 4 * (entire Z depth of the device viewport).

SEE ALSO

DdInqViewport(3D), DdInqPickAperture(3D), DdPickObjs(3D)

NAME

DdSetPickCallback – Set the picking callback object of a device

SYNOPSIS

C:

```
void DdSetPickCallback(device, pickcallbackobj)
DtObject device;
DtObject pickcallbackobj;
```

Fortran:

```
CALL DDSPCB(DEVICE, PKCBOJ)
INTEGER*4 DEVICE
INTEGER*4 PKCBOJ
```

DESCRIPTION

DdSetPickCallback sets the device picking callback object, *pickcallbackobj*, to be used on the device, *device*.

Picking is a method of identifying displayable primitive objects that fall within a specified volume known as a device pick aperture (see *DdSetPickAperture*). A pick is initiated by a call to *DdPickObjs* <*DDPO*>. The pick callback is a callback object that selects which primitive objects found within the pick aperture during a pick (known as "hits") will actually be returned. Doré currently provides three standard pick callback objects that may be used; however, users may also easily provide their own. The standard callback objects are:

DcPickFirst <*DCPKFR*>
: Accept only the first hit found.

DcPickClosest <*DCPKCL*>
: Accept only the hit that was frontmost (closest to the viewer) in the pick aperture.

DcPickAll <*DCPKAL*>
: Add all hits found to the hit list.

Users may want to add other types of callback objects such as those that accept or exclude only hits on certain types of objects or those that accept only up to a maximum number of hits before terminating a search.

The following information is intended mainly for users who wish to create their own pick callback objects.

DdSetPickCallback specifies a callback object that is called each time a hit is detected. That callback gets to peek at each pick path generated by pick hits and decide what to do with the hit. Users create their own pick callback objects by writing a function and making the function into an object using the function, *DoCallback* <*DOCB*>. The function in C should look like this:

```
DtHitStatus
 my_callback_fcn(data, path_elements, path, z_value, view, hits);
DtPtr data;          /* if data is a pointer to data */
Dt32bits data;       /* if data is a value */
DtInt path_elements;
DtInt *path;
DtReal z_value;
DtObject view;
DtInt hits;
```

In Fortran:

```
INTEGER*4 FCN(DATA,PTHELE,PATH,ZVAL,VIEW,HITS)
INTEGER*4 DATA
INTEGER*4 PTHELE
INTEGER*4 PATH
REAL*8 ZVAL
INTEGER*4 VIEW
INTEGER*4 HITS
```

The function takes the following six arguments:

data A *DataVal* or *DataPtr* that contains associated run-time data to be passed to the function (see *DoCallback*).

path_elements
A *DtInt* specifying the number of pick elements in the pick path.

path
A pointer to a *DtInt* that is the beginning of the pick path (described below).

z_value
A *DtReal* specifying the depth of the closest point of the hit (in device coordinates).

view A *DtObject* specifying the view in which the hit was found.

hits A *DtInt* specifying the total number of hits accepted so far during this pick.

A "hit" primitive is uniquely identified by its *pick path*. A pick path is a list of *pick elements* that trace the display tree nodes between the root object and a primitive. Each pick element, for a particular node, consists of three integer values in the display tree:

The object handle for this node's object.

The current pick ID for this node.

The location within this group of the next node. This location is the same as the group element number of the next node in the pick path. Note that the last pick element in the path (or first, if the pick path order is *DcBottomFirst* <DCBOTF>) refers to the primitive that was hit instead of a containing group object. The location of the next node within this primitive object is undefined because this primitive object is the last node (and this primitive object is not a group).

The pick path passed to the callback function will always be ordered from the root node of the group network down to the picked primitive regardless of the current value of the device's pick path order. The function can choose whether to accept this hit and add it to the hit list that will be returned by the current pick, reject the hit, or overwrite the previous hit (path) accepted.

The pick callback function must return a value of type *DtHitStatus*. The valid values are:

DcHitAccept <DCHACC>
DcHitReject <DCHREJ>
DcHitOverwrite <DCHOVW>

Returning *DcHitOverwrite* <DCHOVW> when no hits have yet been accepted behaves the same as *DcHitAccept* <DCHACC>. Any other returned value will result in a *DcPickBadStatus* <DCPKBS> error word being generated and returned by the current call to *DdPickObjs* <DDPO>.

The callback function can choose to stop the execution of the current group or can abort the pick altogether by calling *DsExecutionReturn <DSER>* or *DsExecutionAbort <DSEA>*. An aborted pick will still return the paths of all the hits previously accepted.

Pickability is controlled by the pickability switch attribute. Its value is determined by *DoPickSwitch <DOPS>* and also by namesets and filters (see *DoNameSet* and *DoFilter*). The invisibility attribute is treated similarly. Primitive objects which are invisible or unpickable cannot be picked. By default, all objects are not pickable.

ERRORS

DdSetPickCallback will fail if passed an invalid device handle or pick callback object.

[WARNING - invalid device handle]

[WARNING - invalid pick callback object]

DEFAULTS

The default *DdSetPickCallback* is *DcPickFirst <DCPKFR>*.

SEE ALSO

DdInqPickAperture(3D), DdInqPickCallback(3D), DdPickObjs(3D), DdSetPickAperture(3D), DoFilter(3D), DoNameSet(3D), DoPickID(3D), DoCallback(3D), DsExecutionAbort(3D), DsExecutionReturn(3D)

NAME

DdSetPickPathOrder – Set the order of pick path elements returned to *DdPickObjs*.

SYNOPSIS

C:

```
void DdSetPickPathOrder(device, pathorder)
DtObject device;
DtPickPathOrder pathorder;
```

Fortran:

```
CALL DDSPPO(DEVICE, PTHORD)
INTEGER*4 DEVICE
INTEGER*4 PTHORD
```

DESCRIPTION

DdSetPickPathOrder determines the order of the returned pick path elements when using *DdPickObjs* <*DDPO*> on the given device *device*. See *DdSetPickCallback* for a description of pick paths.

The valid values for the argument *pathorder* are *DcTopFirst* <*DCTOPF*> and *DcBottomFirst* <*DCBOTF*>. *DcTopFirst* <*DCTOPF*> means that the pick path order starts with the largest group in the view containing the primitive object and ends with the picked primitive object itself. *DcBottomFirst* <*DCBOTF*> is the reverse.

Note that the value of the pick path order affects only the order of pick path elements in paths returned from *DdPickObjs* <*DDPO*> and does NOT affect the order of elements as seen by the pick callback functions (see *DdSetPickCallback*).

ERRORS

DdSetPickPathOrder will fail if passed an invalid device handle.

[WARNING - invalid device handle]

DdSetPickCallback will fail if passed an invalid path order.

[WARNING - invalid pick path order]

DEFAULTS

The default pick path order is *DcTopFirst* <*DCTOPF*>.

SEE ALSO

DdInqPickPathOrder(3D), DdPickObjs(3D), DdSetPickCallback(3D)

NAME

DdSetShadeMode – Set the shade mode of a device

SYNOPSIS

C:

```
void DdSetShadeMode(device, mode)
DtObject device;
DtShadeMode mode;
```

Fortran:

```
CALL DDSSM(DEVICE, MODE)
INTEGER*4 DEVICE
INTEGER*4 MODE
```

DESCRIPTION

DdSetShadeMode specifies the shade mode of a device, *device*. The *mode* parameter can take one of the following two values: *DcComponent* <DCCOMP> and *DcRange* <DCRNG>.

Doré can produce output on devices that have limited color capability. These devices are usually referred to as "pseudocolor" devices. Doré always computes shades in full color, and compresses these shades to pseudocolor at output.

The *mode* attribute specifies how this compression is to be performed. This attribute can be either *DcComponent* <DCCOMP> or *DcRange* <DCRNG>. A value of *DcComponent* means that each primary computed color (red, green, and blue) is compressed into a limited bit range and the resulting bit fields are Or'ed together to form a single n-bit value. The compression algorithm is fixed for a specific value of n (for n = 8; 3 bits red, 3 bits green, 2 bits blue).

A value of *DcRange* <DCRNG> specifies that the user has loaded a particular color map into the pseudocolor device which has known ranges of shading (like gold and silver). The *DdInqShadeRanges* <DDQSR> and *DdSetShadeRanges* <DDSSR> calls inquire and setup the contents of specific shade ranges for the device. See these manual pages for further information on this pseudocolor mode.

ERRORS

DdSetShadeMode will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdSetShadeMode will fail if the mode value is invalid.

[WARNING - bad mode value]

DEFAULTS

The default *device* is *DcComponent* <DCCOMP>.

SEE ALSO

DdInqColorEntries(3D), DdInqColorTableSize(3D), DdInqShadeMode(3D), DdInqShadeRanges(3D), DdInqVisualType(3D), DdSetColorEntries(3D), DdSetShadeRanges(3D), DoShadeIndex(3D), DoDevice(3D)

NAME

DdSetShadeRanges – Set one or more shade range table entries of a device

SYNOPSIS

C:

```
void DdSetShadeRanges(device, start, count, entries)
DtObject device;
DtInt start;
DtInt count;
DtInt entries[];
```

Fortran:

```
CALL DDSSR(DEVICE, START, COUNT, ENTRYS)
INTEGER*4 DEVICE
INTEGER*4 START
INTEGER*4 COUNT
INTEGER*4 ENTRYS(COUNT*2)
```

DESCRIPTION

DdSetShadeRanges sets one or more shade range table entries on the device, *device*. The shade range table is used when a device is in pseudocolor mode and the shademode is set to *DcRange* <*DCRNG*>. Each entry in the shade range table consists of a pair of values defining a shade range that specifies a range in the color table entries into which computed intensities map. Intensities between 0.0 and 1.0 are used to linearly interpolate between the two specified limits in the color lookup table. This interpolated color table index is then sent to the device's frame buffer.

The parameter *start* specifies the first entry to be set. The parameter *count* specifies the number of entries to be written. The parameter *entries* is an array containing the new range boundaries.

To use this mode, the application should subdivide the color map into a set of shade ranges. Each color table entry within the range has the same base color (hue) but has steadily increasing intensity as the index goes from the minimum entry in the range to the maximum.

For example, consider an image of a mechanical part consisting of three colors: bronze, silver and gray. Instead of having unused colors in the color map, the application can use shaderanges to partition the color map into the necessary shades of bronze, silver and gray. When a shade is computed by the renderer, it is converted to an intensity between 0.0 and 1.0 that interpolates between the entries of the current shade range (set by *DoShadeIndex* <*DOSI*>). The application program then uses *DdSetShadeRanges* <*DDSSR*> to define three shade ranges:

Shade range 1 specifying the minimum and maximum entries for bronze.

Shade range 2 specifying the minimum and maximum entries for silver.

Shade range 3 specifying the minimum and maximum entries for gray.

The shade index primitive attribute is used to select a particular shade range; see *DoShadeIndex*.

Because the actual size of each color table is specified for each device, an application can adapt the color table to eliminate unneeded entries. The number of entries used by each shade range can be based on the importance of the shade to the image and the size of the devices color lookup table. If the image is being displayed on several devices, each with color lookup tables of different sizes, the number of elements in each shade range can be adjusted appropriately.

ERRORS

DdSetShadeRanges will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdSetShadeRanges will fail if the *start* or *count* parameters refer to entries outside the shade range table boundries.

[WARNING - bad start and/or count values]

SEE ALSO

DdInqColorEntries(3D), DdInqColorTableSize(3D), DdInqShadeMode(3D), DdInqShadeRanges(3D), DdInqVisualType(3D), DdSetColorEntries(3D), DdSetShadeMode(3D), DoShadeIndex(3D)

NAME

DdSetViewport – Define a device viewport for a device

SYNOPSIS

C:

```
void DdSetViewport(device, viewport)
DtObject device;
DtVolume *viewport;
```

Fortran:

```
CALL DDSDV(DEVICE, VWPORT)
INTEGER*4 DEVICE
REAL*8 VWPORT(6)
```

DESCRIPTION

DdSetViewport specifies the device viewport for the indicated device, *device*. The device viewport specifies the portion of the device's extent onto which the frame boundary is mapped (i.e. the portion of the device's view surface to be used for display). The default viewport uses all of the current device volume. If the device extent changes (for example, if an X-window is resized), then the default viewport automatically changes to fit the new device extent, unless the device is specified as a non-auto-resizing device (see *DoDevice*).

ERRORS

DdSetViewport will fail if the device handle is invalid.

[WARNING - invalid device handle]

SEE ALSO

DdInqExtent(3D), DdInqViewport(3D), DoDevice(3D)

NAME

DdUpdate – Update the specified device

SYNOPSIS

C:

```
void DdUpdate(device)
DtObject device;
```

Fortran:

```
CALL DDU(DEVICE)
INTEGER*4 DEVICE
```

DESCRIPTION

DdUpdate causes the indicated device *device* to update itself. If the device has no frame attached to it, this operation does nothing. If the device has a frame attached to it, all views attached to that frame will be updated.

ERRORS

DdUpdate will fail if the device handle is invalid.

[WARNING - invalid device handle]

DdUpdate will fail if an update is in progress.

[WARNING - traversal already in progress]

SEE ALSO

DfUpdate(3D), DvSetUpdateType(3D) DvUpdate(3D)

NAME

DEDOD – Deallocate space used by the private data of an object of a user-defined class (Fortran only)

SYNOPSIS

Fortran:

```
DEDOD(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DEDOD deallocates the space used by the private data of the object *OBJECT*. *DEDOD* is used by the destroy method of a Fortran user-defined primitive to delete the private data of the object with which it is called.

SEE ALSO

DeCreateObject(3D), DEROD(3D), DEWOD(3D)

NAME

DEROD – Read private data of an object of a user-defined class (Fortran only)

SYNOPSIS

Fortran:

```
DEROD(OBJECT, TODATA, SIZE)
INTEGER*4 OBJECT
REAL*8 TODATA(*)
INTEGER*4 SIZE
```

DESCRIPTION

DEROD is used by the method routines of a Fortran user-defined primitive to get the private data of the object with which the method is called. It copies *SIZE* bytes from the private data of the object *OBJECT* to *TODATA*

Note that the type of *TODATA* is not important since *DEROD* only uses the address of it as a starting point for the copy. Typically, *TODATA* will be an array containing variables of different types (via an EQUIVALENCE statement).

SEE ALSO

DeCreateObject(3D), DEWOD(3D), DEDOD(3D)

NAME

DEWOD – Write private data of an object of a user-defined class (Fortran only)

SYNOPSIS

Fortran:

```
DEWOD(OBJECT, FROMDATA, SIZE)
INTEGER*4 OBJECT
REAL*8 TODATA(*)
INTEGER*4 SIZE
```

DESCRIPTION

DEWOD is used by the method routines of Fortran user-defined primitives to write to the private data of the object with which the method is called. It copies *SIZE* bytes from *FROMDATA* to the private data of the object *OBJECT*.

Note that the type of *FROMDATA* is not important since *DEWOD* only uses the address of it as a starting point for the copy. Typically, *FROMDATA* will be an array containing variables of different types (via an EQUIVALENCE statement).

SEE ALSO

DeCreateObject(3D), DEROD(3D), DEDOD(3D)

NAME

DeAddClass – Add a new class (object type)

SYNOPSIS

C:

```
DtInt DeAddClass(name, count, list, default_routine)
DtPtr name;
DtInt count;
DtInt list[];
DtPFI default_routine;
```

Fortran:

```
INTEGER*4 DEAC(NAME, N, COUNT, LIST, DFLTRT)
CHARACTER*N NAME
INTEGER*4 COUNT
INTEGER*4 LIST(*)
EXTERNAL DFLTRT

DEAMTH(LIST, METHOD, RTN, MTHCNT)
INTEGER*4 LIST(*)
INTEGER*4 METHOD
EXTERNAL RTN
INTEGER*4 MTHCNT
```

DESCRIPTION

DeAddClass adds a new class (object type) to the Doré system. It returns a unique class identification number.

Each class in Doré must have a unique name. The parameter *name* specifies the name for the new class. The convention is to name the class the same as the routine for creating an object of the new class.

The parameter *list* is a list of Doré symbolic method names, and pointers to user-provided routines to be used as those methods in the new class. Elements 0, 2, 4, etc. are method names. Elements 1, 3, 5, etc. are the pointers to the function names of the user-provided routines. Routine *list*[2n+1] is associated with method *list*[2n]. The parameter *count* specifies the number of methods in *list*.

The parameter *default_routine* is a pointer to a routine that will be used for methods that are not specified in *list*, or *DcNullPtr <DCNULL>*.

Methods used by user-defined primitives are:

DcMethodCmpBndVolume <DCMCBV>
Compute bounding volume box.

DcMethodDestroy <DCMDST>
Deallocate space used by object.

DcMethodDynRender <DCMDR>
Dynamic rendering.

DcMethodGlbrndIniObjs <DCMGIO>
Production rendering.

DcMethodPick <DCMPCK>
Determine if object has been picked.

DcMethodPrint <DCMPRT>
Print object information.

DcMethodStdRenderDisplay <DCMSRD>
Update and execute alternate object.

DcMethodUpdStdAltObj <DCMSAO>
Update alternate object and return.

FORTRAN SPECIFIC

The parameter *NAME* is a string of length *N* representing the name of the new class.

In Fortran the method list is built using *DEAMTH*. The calling routine must allocate space for the list (twice as many elements as methods to be inserted). *DEAMTH* is called once for every method/routine pair to be added to the list. The parameter *MTHCNT* is the number of methods that have already been added to the list.

ERRORS

DeAddClass will fail if a method in *list* is invalid.

[WARNING - invalid method]

NAME

DeCreateObject – Create an internal Doré object

SYNOPSIS

C:

```
DtObject DeCreateObject(class_id, object_data)
DtInt class_id;
DtPtr object_data;
```

Fortran:

```
INTEGER*4 DECO(CLASID,DATA,SIZE)
INTEGER*4 CLASID
REAL*8 DATA(*)
INTEGER*4 SIZE
```

DESCRIPTION

DeCreateObject creates an internal Doré object, and returns the object handle.

The parameter *class_id* is the unique class identifier for the object to be created. For user-defined classes this will be the class identification number returned by *DeAddClass* <*DEAC*>.

C SPECIFIC

The C version of *DeCreateObject* assumes that the calling routine has already allocated space for the private data for the object, and that *object_data* points to this data.

FORTRAN SPECIFIC

The Fortran function, *DECO*, assumes that space for the private data has *not* been allocated. It will allocate *SIZE* bytes, and copy the same number of bytes from *DATA* to the newly allocated space. This means that the method routines of a user-defined primitive implemented in Fortran do not have direct access to the private data of the objects. The private data of those objects can be accessed from Fortran through calls to *DEROD*, *DEWOD*, and *DEDOD*.

Note that the type of *DATA* is not important since *DECO* only uses the address of it as a starting point for the copy. Typically, *DATA* will be an array containing variables of different types (via an EQUIVALENCE statement).

ERRORS

DeCreateObject will fail if the class identifier is invalid.

[WARNING - invalid class handle]

SEE ALSO

DeAddClass(3D), DEDOD(3D), DEROD(3D) DEWOD(3D)

NAME

DeDeleteObject – Delete a Doré object

SYNOPSIS

C:

```
void DeDeleteObject(object)
DtObject object;
```

Fortran:

```
CALL DEDO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DeDeleteObject deletes the Doré object, *object*, if it is not referenced by other objects, and if the user has not placed a hold on the object with *DsHoldObj() <DSHO>*.

ERRORS

DeDeleteObject will fail if the object handle is invalid.

[WARNING - invalid or deleted object]

SEE ALSO

DsHoldObj(3D), DsReleaseObj(3D)

NAME

DeExecuteAlternate – Execute the current method on an alternate object

SYNOPSIS

C:

```
void DeExecuteAlternate(object)
DtObject object;
```

Fortran:

```
CALL DEEA(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DeExecuteAlternate executes the current method on the object, *object*. It is assumed that the object is an alternate object for a user-defined primitive.

ERRORS

DeExecuteAlternate will fail if the object handle is invalid.

[WARNING - invalid or deleted object]

NAME

DeInitializeObjPick – Initialize picking for an object

SYNOPSIS

C:

```
void DeInitializeObjPick(object)
DtObject object;
```

Fortran:

```
CALL DEIOP(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DeInitializeObjPick is used be the picking method of a user-defined primitive to initialize picking for the object, *object*, of the user-defined class.

ERRORS

DeInitializeObjPick will fail if the object handle is invalid.

[WARNING - invalid or deleted object]

SEE ALSO

DeAddClass(3D)

NAME

DeInqPickable – Return whether a class is pickable

SYNOPSIS

C:

```
DtFlag DeInqPickable(class_id)
DtInt class_id;
```

Fortran:

```
INTEGER*4 DEQP(CLSID)
INTEGER*4 CLSID
```

DESCRIPTION

DeInqPickable returns *DcTrue* <*DCTRUE*> if objects of type *class_id* are currently pickable and executable. Otherwise *DcFalse* <*DCFALS*> is returned. For user-defined primitives, *class_id* is the unique class identifier returned by *DeAddClass* <*DEAC*>. *DeInqPickable* is used by the pick method of user-defined primitive classes.

ERRORS

DeInqPickable will fail if the class identifier is invalid.

[WARNING - invalid class handle]

SEE ALSO

DeAddClass(3D)

NAME

DeInqRenderable – Return whether a class is renderable

SYNOPSIS

C:

```
DtFlag DeInqRenderable(class_id)
DtInt class_id;
```

Fortran:

```
INTEGER*4 DEQR(CLSID)
INTEGER*4 CLSID
```

DESCRIPTION

DeInqRenderable returns *DcTrue* <*DCTRUE*> if objects of type *class_id* are currently executable and not invisible. Otherwise *DcFalse* <*DCFALS*> is returned. For user-defined primitives, *class_id* is the unique class identifier returned by *DeAddClass* <*DEAC*>. *DeInqRenderable* is used by the render methods of user-defined classes.

ERRORS

DeInqRenderable will fail if the class identifier is invalid.

[WARNING - invalid class handle]

SEE ALSO

DeAddClass(3D)

NAME

DfInqBoundary – Return the frame boundary

SYNOPSIS

C:

```
void DfInqBoundary(frame, boundary)
DtObject frame;
DtVolume *boundary;
```

Fortran:

```
CALL DFQB(FRAME, BNDRY)
INTEGER*4 FRAME
REAL*8 BNDRY(6)
```

DESCRIPTION

DfInqBoundary queries the boundary of a specified frame, *frame*. The frame is mapped onto the largest right rectangular volume that can fit within the device viewport such that aspect ratio in X and Y is preserved and that the Z extent of the frame is mapped to the entire Z extent of the device.

When this mapping is performed, there may be extra "white space" inside the device viewport. The function *DfSetJust* <*DFSJ*> can be used to position this white space.

ERRORS

DfSetBoundary will fail if the frame handle is invalid.

[WARNING - invalid frame handle]

SEE ALSO

DdSetFrame(3D), DfSetBoundary(3D), DfSetJust(3D), DfUpdate(3D)

NAME

DfInqJust – Return the frame justification

SYNOPSIS

C:

```
void DfInqJust(frame, left, bottom)
DtObject frame;
DtReal *left, *bottom ;
```

Fortran:

```
CALL DFQJ(FRAME, LEFT, BOTTOM)
INTEGER*4 FRAME
REAL*8 LEFT, *BOTTOM
```

DESCRIPTION

DfInqJust queries the justification of a specified frame, *frame*. When the aspect ratios of the frame border and the device viewport are not the same, there will be extra "white space" inside the device viewport. The frame justification provides a mechanism for controlling where this white space is positioned.

The parameter *left* defines a value between 0.0 and 1.0 inclusive specifying the fraction of the white space to be inserted between the bottom left hand corner of the viewport and the left of the displayed image. Similarly, the parameter *bottom* specifies the fraction of whitespace to be inserted between the bottom left corner of the viewport and the bottom of the displayed image.

ERRORS

DfInqJust will fail if the frame handle is invalid.

[WARNING - invalid frame handle]

SEE ALSO

DdSetFrame(3D), DfSetJust(3D), DfSetBoundary(3D), DfUpdate(3D)

NAME

DfInqViewGroup – Return the handle for a frame's view group

SYNOPSIS

C:

```
DtObject DfInqViewGroup(frame)
DtObject frame;
```

Fortran:

```
INTEGER*4 DFQVG(FRAME)
INTEGER*4 FRAME
```

DESCRIPTION

DfInqViewGroup returns the handle for a frame's view group. The view group is part of every frame and contains the view handles of all the views associated with the frame (see *DoView*). When the frame is updated, all views in its view group are updated in order. One may edit the view group (via the group editing functions) to arrange the views in the desired order of update. When a frame is updated, all non-view objects found in its view group are ignored.

ERRORS

DfInqViewGroup will fail if the frame handle is invalid; *DcNullObject* <DCNULL> is returned.

[WARNING - invalid frame handle]

SEE ALSO

DoView(3D)

NAME

DfSetBoundary – Set the frame boundary

SYNOPSIS

C:

```
void DfSetBoundary(frame, boundary)
DtObject frame;
DtVolume *boundary;
```

Fortran:

```
CALL DFSB(FRAME, BNDRY)
INTEGER*4 FRAME
REAL*8 BNDRY(6)
```

DESCRIPTION

DfSetBoundary sets the boundary of a specified frame, *frame*. A frame defines a device-independent 3D local coordinate system within which one or more views can be positioned. The frame boundary can be queried using *DfInqBoundary* <*DFQB*>.

The frame is mapped onto the largest right rectangular volume that can fit within the device viewport such that aspect ratio in X and Y is preserved and that the Z extent of the frame is mapped to the entire Z extent of the device. When this mapping is performed, there may be extra "white space" inside the device viewport. The function *DfSetJust* <*DFSJ*> can be used to position this white space.

ERRORS

DfSetBoundary will fail if the frame handle is invalid.

[WARNING - invalid frame handle]

DEFAULTS

The default frame boundary extends from (0.0, 0.0, 0.0) to (1.0, 1.0, 1.0).

SEE ALSO

DdSetFrame(3D), DfInqBoundary(3D), DfSetJust(3D), DfUpdate(3D)

NAME

DfSetJust – Set the frame justification

SYNOPSIS

C:

```
void DfSetJust(frame, left, bottom)
DtObject frame;
DtReal left, bottom;
```

Fortran:

```
CALL DFSJ(FRAME, LEFT, BOTTOM)
INTEGER*4 FRAME
REAL*8 LEFT, BOTTOM
```

DESCRIPTION

DfSetJust sets the justification of a specified frame, *frame*. When the aspect ratios of the frame border and the device viewport are not the same, there will be extra "white space" inside the device viewport. The frame justification provides a mechanism for controlling where this white space is positioned.

The *left* parameter defines a value between 0.0 and 1.0 inclusive specifying the fraction of the white space to be inserted between the bottom left hand corner of the viewport and the left of the displayed image. Similarly, the *bottom* parameter specifies the fraction of white space to be inserted between the bottom left hand corner of the viewport and the bottom of the displayed image.

ERRORS

DfSetJust will fail if the frame handle is invalid.

[WARNING - invalid frame handle]

DfSetJust will fail if the *left* or *bottom* parameters are out of range.

[WARNING - value out of range]

DEFAULTS

The default is *left* = 0.5 and *bottom* = 0.5 creating an even distribution of white space in both directions.

SEE ALSO

DdSetFrame(3D), DfInqBoundary(3D), DfInqJust(3D), DfUpdate(3D)

NAME

DfUpdate – Update the specified frame

SYNOPSIS

C:

```
void DfUpdate(frame)
DtObject frame;
```

Fortran:

```
CALL DFU(FRAME)
INTEGER*4 FRAME
```

DESCRIPTION

DfUpdate causes the frame, *frame*, to update itself on all of the devices to which it is attached.

ERRORS

DfUpdate will fail if the frame handle is invalid.

[WARNING - invalid frame handle]

DfUpdate will fail if an update is in progress.

[WARNING - traversal already in progress]

SEE ALSO

DoFrame(3D), DvUpdate(3D)

NAME

DgAddObj – Add an object to the currently open group

SYNOPSIS

C:

```
void DgAddObj(object)
DtObject object;
```

Fortran:

```
CALL DGAO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DgAddObj adds an object, *object*, to the currently open group. The object is inserted into the group at the position indicated by the element pointer, with the element pointer and all subsequent following elements moving forward one position. This results in the element pointer's pointing after the object just inserted. *DgReplaceObj <DGRO>* is used to replace the object specified by the group's element pointer.

The most common way to call the *DgAddObj* function is:

```
DgAddObj(DoXXXX());
```

where the *DoXXXX()* call creates and returns an object of type XXXX. The object is added to the currently open group through the *DgAddObj* call. One can also create an object through a separate call to a *DoXXXX* routine and then call the *DgAddObj* routine with the object directly:

```
object = DoXXXX();
DsHoldObj(object);
DgAddObj(object);
```

One may also call this function in the following manner:

```
DgAddObj(DsHoldObj(object=DoXXXX()));
```

This will result in the *object* variable's getting the value of the newly created object as well as placing the object in the currently open group.

The call to *DsHoldObj <DSHO>* is needed only if the user also wants to use the object created for other purposes. Note that any object held with a *DsHoldObj <DSHO>* will not be deleted. The application program is always responsible for reclamation of storage from obsolete held objects.

ERRORS

DgAddObj will fail if *object* is not a valid object.

[WARNING - invalid or deleted object]

DgAddObj will fail if no group is currently open.

[WARNING - a group is not currently open]

SEE ALSO

DgAddObjToGroup(3D), DgInqOpen(3D), DgOpen(3D), DgReplaceObj(3D), DoGroup(3D), DsHoldObj(3D)

NAME

DgAddObjToGroup – Add an object to a specified group

SYNOPSIS

C:

```
void DgAddObjToGroup(group,object)
DtObject group;
DtObject object;
```

Fortran:

```
CALL DGAOG(GROUP,OBJECT)
INTEGER*4 GROUP
INTEGER*4 OBJECT
```

DESCRIPTION

DgAddObjToGroup adds an object, *object*, to the specified group, *group*. The object is inserted into the group at the position indicated by the element pointer, and the element pointer moves forward one position so that it points after the object just inserted. Any objects that follow the element pointer also move forward.

The most common way to call the *DgAddObjToGroup* function is:

```
DgAddObjToGroup(group,DoXXXX());
```

where the *DoXXXX()* call creates and returns an object of type XXXX. The object is added to the designated group through the *DgAddObjToGroup* call. One can also create an object through a separate call to a *DoXXXX* routine and then call the *DgAddObjToGroup* routine with the object directly:

```
object = DoXXXX();
DsHoldObj(object);
DgAddObjToGroup(group,object);
```

One may also call this function in the following manner:

```
DgAddObjToGroup(group,DsHoldObj(object=DoXXXX()));
```

This causes *object* to take the value of the newly created object and places the object in the specified group.

The call to *DsHoldObj* is needed only if the user also wants to use the created object for other purposes. Any object held with *DsHoldObj* will not be deleted. The application program is always responsible for reclamation of storage from obsolete held objects.

ERRORS

DgAddObjToGroup will fail if called with an invalid group.

[WARNING - invalid group handle]

DgAddObjToGroup will fail if called with an invalid object.

[WARNING - invalid or deleted object]

SEE ALSO

DgAddObj(3D), DgInqOpen(3D), DgOpen(3D), DoGroup(3D), DsHoldObj(3D)

NAME

DgCheck – Check for circularities within a group network

SYNOPSIS

C:

```
DtGroupNetworkStatus DgCheck(group)
DtObject group;
```

Fortran:

```
INTEGER*4 DGCK(GROUP)
INTEGER*4 GROUP
```

DESCRIPTION

DgCheck finds out if there are any circularities within the group network referenced by the root node, *group*. Many execution methods (i.e., for rendering and picking) do not check for the existence of circularity within the group networks they execute and will develop fatal errors if such circularities are allowed to persist. It is the user's responsibility never to use circular group networks in Doré.

The return value will be one of the following:

DcGroupOk <DCGOK>
The group is free of circularities.

DcGroupBad <DCGBAD>
The group has a non-recoverable circularity.

ERRORS

DgCheck will fail if the group handle is invalid.

[WARNING - invalid group handle]

DgCheck will fail if an update is in progress.

[WARNING - traversal in progress]

SEE ALSO

DgOpen(3D), DoGroup(3D)

NAME

DgClose – Close a group object

SYNOPSIS

C:

```
DtObject DgClose()
```

Fortran:

```
INTEGER*4 DGCS()
```

DESCRIPTION

DgClose closes the currently open group and returns its object handle. If no group is currently open, then *DcNullObject* <*DCNULL*> is returned. Open groups may be nested in Doré. If there are nested open groups, *DgClose* closes only the most recently opened group (also known as the active group). Only one group may be active at any one time. Groups are opened with *DgOpen* <*DGO*> or created open using the appropriate arguments to *DoGroup* <*DOG*>.

SEE ALSO

DgInqOpen(3D), DgOpen(3D), DoGroup(3D)

NAME

DgDelEle – Remove elements from the currently open group

SYNOPSIS

C:

```
void DgDelEle(count)
DtInt count;
```

Fortran:

```
CALL DGDE(COUNT)
INTEGER*4 COUNT
```

DESCRIPTION

DgDelEle deletes *count* number of elements from the currently open group. The deletion starts at the position of the element pointer in the currently open group. If its position in the group plus *count* is larger than the total number of elements in the group, all elements from the element pointer position to the end of the group are deleted. If the element pointer position plus *count* is smaller than the total number of elements in the group, the element pointer points to the element positioned after the last one that was deleted.

ERRORS

DgDelEle will fail if no group is currently open.

[WARNING - a group is not currently open]

SEE ALSO

DgDelEleBetweenLabels(3D), DgDelEleRange(3D), DgEmpty(3D), DgInqElePtr(3D), DgSetElePtr(3D)

NAME

DgDelEleBetweenLabels – Remove all elements between labels from the currently open group

SYNOPSIS

C:

```
DtFlag DgDelEleBetweenLabels(label1, label2)
DtInt label1;
DtInt label2;
```

Fortran:

```
INTEGER*4 DGDEL(LABEL1, LABEL2)
INTEGER*4 LABEL1
INTEGER*4 LABEL2
```

DESCRIPTION

DgDelEleBetweenLabels deletes all of the elements between two labels, *label1* and *label2*, from the currently open group. Labels are elements that can be placed anywhere within a group to mark a special location. There may be more than one label with the same value.

DgDelEleBetweenLabels searches forword from the current position, for the first occurrence of *label1*. If found, a search is then made from that point for the first occurrence of *label2*. If both labels are found, then *DgDelEleBetweenLabels* deletes all the elements between the labels but not the labels themselves. After the deletion, the element pointer will end up pointing to the second label and *DcTrue* <*DCTRUE*> will be returned. If neither label is found, *DcFalse* <*DCFALS*> will be returned.

ERRORS

DgDelEleBetweenLabels will fail if no group is currently open. *DcFalse* <*DCFALS*> will be returned.

[WARNING - a group is not currently open]

DgDelEleBetweenLabels will fail if passed an invalid label.

[WARNING - can't find first label]

[WARNING - can't find second label]

SEE ALSO

DgDelEle(3D), DgDelEleRange(3D), DgEmpty(3D)

NAME

DgDelEleRange – Remove a range of elements from the currently open group

SYNOPSIS

C:

```
void DgDelEleRange(from, to)
DtInt from;
DtInt to;
```

Fortran:

```
CALL DGDER(FROM, TO)
INTEGER*4 FROM
INTEGER*4 TO
```

DESCRIPTION

DgDelEleRange deletes a range of elements between the specified locations, *from* and *to*, from the currently open group. If the *from* location is negative, elements will be deleted starting at the beginning of the group. If the *to* location is past the end of the group, elements will be deleted only to the end of the group. The element pointer will point to the element after the last one that was deleted.

ERRORS

DgDelEleRange will fail if no group is currently open.

[WARNING - a group is not currently open]

SEE ALSO

DgDelEle(3D), DgDelEleBetweenLabels(3D), DgEmpty(3D)

NAME

DgEmpty – Remove all elements from a specified group

SYNOPSIS

C:

```
void DgEmpty(group)
DtObject group;
```

Fortran:

```
CALL DGE(GROUP)
INTEGER*4 GROUP
```

DESCRIPTION

DgEmpty deletes all elements from the group, *group*.

ERRORS

DgEmpty will fail if the group handle is invalid.

[WARNING - invalid group handle]

SEE ALSO

DgDelEle(3D), DgDelEleBetweenLabels(3D), DgDelEleRange(3D), DoGroup(3D)

NAME

DgInqElePtr – Return the location of the group element pointer of the current group

SYNOPSIS

C:

```
DtInt DgInqElePtr()
```

Fortran:

```
INTEGER*4 DGQEP()
```

DESCRIPTION

DgInqElePtr returns the current location of the group element pointer of the currently open group.

Each group has an element pointer that identifies the current editing position within the group. The element pointer points between elements of a group. A value of 0 indicates that the element pointer is pointing before the first element in the group, and a value of *n* (where *n* is the current size of the group) indicates that the element pointer is pointing after the last element in the group.

ERRORS

DgInqElePtr will fail if no group is currently open; the value -1 is returned.

[WARNING - a group is not currently open]

SEE ALSO

DgSetElePtr(3D), DgSetElePtrRelLabel(3D)

NAME

DgInqObjAtPos – Return the object at a specified position in a group

SYNOPSIS

C:

```
DtObject DgInqObjAtPos(group, offset, position_orientation)
DtObject group ;
DtInt offset;
DtRelPosition position_orientation;
```

Fortran:

```
INTEGER*4 DGQOP(GROUP, OFFSET, POSORT)
INTEGER*4 GROUP
INTEGER*4 OFFSET
INTEGER*4 POSORT
```

DESCRIPTION

DgInqObjAtPos returns the object at the position, *offset*, of position orientation, *position_orientation*, within group, *group*. Positions within a group are zero based with element 0 being the first element in the group and element *n-1* being the last element in the group of size *n*.

The *position_orientation* parameter can take one of the following values:

DcBeginning <DCBEG>
The element pointer is positioned relative to the beginning of the group.

DcEnd <DCEND>
The element pointer is positioned relative to the end of the group.

DcCurrent <DCCUR>
The element pointer is positioned relative to the element pointer's current position.

If no object is at the specified position, then *DcNullObject <DCNULL>* is returned. Otherwise, the object at the specified position is returned.

ERRORS

DgInqObjAtPos will fail if called with an invalid group; the value *DcNullObject <DCNULL>* is returned.

[WARNING - invalid group handle]

NAME

DgInqOpen – Determine which, if any, group is open

SYNOPSIS

C:

DtObject DgInqOpen()

Fortran:

INTEGER*4 DGQO()

DESCRIPTION

DgInqOpen finds out which group, if any, is currently open. If more than one group is open, *DgInqOpen* returns the handle of the most recently opened group (the active group). Note that there can be at most one group active at any given time. Most group editing operations are performed on the active group.

DgInqOpen will return the group name of the currently open group if a group is currently open. If no group is currently open, *DgInqOpen* will return *DcNullObject* <*DCNULL*>.

DgInqOpen will always work.

SEE ALSO

DgOpen(3D), DoGroup(3D)

NAME

DgInqSize – Return the number of elements in the specified group

SYNOPSIS

C:

```
DtInt DgInqSize(group)
DtObject group;
```

Fortran:

```
INTEGER*4 DGQS(GROUP)
INTEGER*4 GROUP
```

DESCRIPTION

DgInqSize returns the total number of elements in the group, *group*.

ERRORS

DgInqSize will fail if the group is invalid; the value -1 will be returned.

[WARNING - invalid group handle]

SEE ALSO

DoGroup(3D), DoInLineGroup(3D)

NAME

DgOpen – Open a group object

SYNOPSIS

C:

```
void DgOpen(group, append)
DtObject group;
DtFlag append;
```

Fortran:

```
CALL DGO(GROUP, APPEND)
INTEGER*4 GROUP
INTEGER*4 APPEND
```

DESCRIPTION

DgOpen opens the group object, *group*. Open groups can be nested in Doré, but there can be only one active group at any given time. The active group is the most recently opened group. It is affected by any implicit *DgXXXX* commands, such as *DgReplaceObj <DGRO>* and *DgAddObj <DGAO>*.

If the append flag, *append*, is *DcFalse <DCFALS>*, the group's element pointer is left as it was before the group was opened. If the append flag is *DcTrue <DCTRUE>*, the group element pointer is placed at the end of the group so that new elements are appended to the open group.

ERRORS

DgOpen will fail if the group handle is invalid.

[WARNING - invalid group handle]

SEE ALSO

DgInqOpen(3D), DoGroup(3D), DoInLineGroup(3D)

NAME

DgReplaceObj – Replace an object in the currently open group

SYNOPSIS

C:

```
void DgReplaceObj(object)
DtObject object;
```

Fortran:

```
CALL DGRO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DgReplaceObj places the object, *object*, in the currently open group. If the group's element pointer is not at the end of the group, the object *replaces* the object immediately following the group's element pointer. If the group's element pointer is after the last element in the group, the object is *appended* to the end of the group. In both cases, the group element pointer remains pointing at the object just replaced or inserted; i.e., the element pointer never moves.

The most common way to call the *DgReplaceObj* function is:

```
DgReplaceObj(DoXXXX());
```

where the *DoXXXX()* call creates and returns an object of type XXXX. The object is added to the currently open group through the *DgReplaceObj* call. One can also create an object through a separate call to a *DoXXXX* routine and then call the *DgReplaceObj* routine with the object directly:

```
object = DoXXXX();
DsHoldObj(object);
DgReplaceObj(object);
```

One may also call this function in the following manner:

```
DgReplaceObj(DsHoldObj(object=DoXXXX()));
```

This will place the handle of the newly created object in *object* and place the object in the currently open group.

The call to *DsHoldObj <DSHO>* is needed only if the user also wants to use the object created for other purposes. Note that any object held with a *DsHoldObj <DSHO>* will not be deleted. The application program is always responsible for reclamation of storage from held objects that are no longer needed.

ERRORS

DgReplaceObj will fail if passed an invalid object.

[WARNING - invalid or deleted object]

DgReplaceObj will fail if no group is currently open.

[WARNING - a group is not currently open]

SEE ALSO

DgAddObjToGroup(3D), DgInqOpen(3D), DgOpen(3D), DoGroup(3D), DsHoldObj(3D)

NAME

DgReplaceObjInGroup – Replace an object in a specified group

SYNOPSIS

C:

```
void DgReplaceObjInGroup(group, object)
DtObject group;
DtObject object;
```

Fortran:

```
CALL DGROG(GROUP, OBJECT)
INTEGER*4 GROUP
INTEGER*4 OBJECT
```

DESCRIPTION

DgReplaceObjInGroup places the object, *object*, in the specified group, *group*. If the group's element pointer is not at the end of the group, the object *replaces* the object immediately following the group's element pointer. If the group's element pointer is after the last element in the group, the object is *appended* to the end of the group. In both cases the element pointer remains pointing at the element just replaced or inserted; i.e., the element pointer never moves.

The most common way to call the *DgReplaceObjInGroup* function is:

```
DgReplaceObjInGroup(group, DoXXXX());
```

where the *DoXXXX()* call creates and returns an object of type XXXX. The object is added to *group* through the *DgReplaceObjInGroup* call. One can also create an object through a separate call to a *DoXXXX* routine and then call the *DgReplaceObjInGroup* routine with the object directly:

```
object = DoXXXX();
DsHoldObj(object);
DgReplaceObjInGroup(group, object);
```

One may also call this function in the following manner:

```
DgReplaceObjInGroup(group, DsHoldObj(object=DoXXXX()));
```

object would then contain a handle to the newly created object and the object would be a member of *group*.

The call to *DsHoldObj* is needed only if the user also wants to use the object created for other purposes. Note that any object held with *DsHoldObj* <*DSHO*> will not be deleted. The application program is always responsible for reclamation of storage from obsolete held objects.

ERRORS

DgReplaceObjInGroup will fail if the group handle is invalid.

[WARNING - invalid group handle]

DgReplaceObjInGroup will fail if passed an invalid object.

[WARNING - invalid or deleted object]

SEE ALSO

DgInqOpen(3D), DgOpen(3D), DgReplaceObj(3D), DoGroup(3D), DsHoldObj(3D)

NAME

DgSetElePtr – Set the group element pointer within the current group

SYNOPSIS

C:

```
void DgSetElePtr(element_ptr, position_orientation)
DtInt element_ptr;
DtRelPosition position_orientation;
```

Fortran:

```
CALL DGSEP(ELEPTR, POSORT)
INTEGER*4 ELEPTR
INTEGER*4 POSORT
```

DESCRIPTION

DgSetElePtr sets the group element pointer to the location, *element_ptr*, of position orientation, *position_orientation*, in the currently open group.

Each group has an element pointer that identifies the current editing position within the group. The element pointer points between elements of a group. A value of 0 indicates that the element pointer is pointing before the first element in the group. A value of *n*, where *n* is the current size of the group, indicates that the element pointer is pointing after the last element in the group.

The *position_orientation* parameter takes one of the following values:

DcBeginning <DCBEG>
The element pointer is positioned relative to the begining of the group.

DcEnd <DCEND>
The element pointer is positioned relative to the end of the group.

DcCurrent <DCCUR>
The element pointer is positioned relative to the element pointer's current position.

ERRORS

DgSetElePtr will fail if no group is currently open.

[WARNING - a group is not currently open]

SEE ALSO

DgInqElePtr(3D), DgSetElePtrRelLabel(3D)

NAME

DgSetElePtrRelLabel – Set the group element pointer relative to a label

SYNOPSIS

C:

```
DtFlag DgSetElePtrRelLabel(label, offset)
DtInt label;
DtInt offset;
```

Fortran:

```
INTEGER*4 DGSEPL(LABEL, OFFSET)
INTEGER*4 LABEL
INTEGER*4 OFFSET
```

DESCRIPTION

DgSetElePtrRelLabel sets the group element pointer to a position relative to the label, *label*, within the current open group. Labels are elements that can be placed anywhere within a group to mark a special location. There can be more than one label with the same value.

DgSetElePtrRelLabel searches for the first occurrence of *label* starting from the current element pointer position. If no occurrence of the specified label exists between the element pointer and the end of the open group, *DcFalse* <*DCFALS*> is returned and the element pointer is left unchanged. If the label is found, *offset* (either a positive or negative integer) is added to the label's location to determine the new current group element pointer location. Locations less than zero are set to zero; locations beyond the end of the group are set to the end of the group. *DcTrue* <*DCTRUE*> is returned on completion.

ERRORS

DgSetElePtrRelLabel will fail if no group is currently open.

[WARNING - a group is not currently open]

SEE ALSO

DgInqElePtr(3D), DgSetElePtr(3D), DgSetElePtrRelLabel(3D), DoLabel(3D)

NAME

DoAmbientIntens – Create an ambient intensity primitive attribute object

SYNOPSIS

C:

```
DtObject DoAmbientIntens(intensity)
DtReal intensity;
```

Fortran:

```
INTEGER*4 DOAMBI(INTENS)
REAL*8 INTENS
```

DESCRIPTION

DoAmbientIntens creates an ambient intensity primitive attribute object. The *intensity* parameter is used to specify the intensity of a surface's ambient response to light from ambient light sources in the environment. The ambient intensity normally ranges from 0.0 to 1.0 signifying the contribution of ambient component to the overall shade of the object's surface.

The ambient intensity is one of three aspects of the ambient component of a surface. The other two components are the *DoDiffuseColor* <*DODIFC*> and the *DoAmbientSwitch* <*DOAMBS*>.

DEFAULTS

The default for *DoAmbientIntens* is 0.3.

SEE ALSO

DoDiffuseColor(3D), DoAmbientSwitch(3D)

NAME

DoAmbientSwitch – Create an ambient switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoAmbientSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOAMBS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoAmbientSwitch creates an ambient switch primitive attribute object. The parameter *switch* specifies whether or not the surfaces of subsequently executed primitive objects have an ambient component contribution.

The ambient component of an object's surface shade will be computed using the diffuse color (see *DoDiffuseColor*) and the ambient intensity (see *DoAmbientIntens*) attributes to determine the response of the surface to ambient light.

If the ambient switch attribute of an object is *DcOff* <*DCOFF*>, that object will render itself without an ambient shading component regardless of the settings of the diffuse color and ambient intensity attributes. The ambient component of the surface will not be computed.

DEFAULTS

The default value for *DoAmbientSwitch* is *DcOn* <*DCON*>.

SEE ALSO

DoDiffuseColor(3D), DoAmbientIntens(3D)

NAME

DoAnnoText – Create an annotation text primitive object

SYNOPSIS

C:

```
DtObject DoAnnoText(position, string)
DtReal position[3];
DtPtr string;
```

Fortran:

```
INTEGER*4 DOANNT(POSITN, STRING, N)
INTEGER*4 N
REAL*8 POSITN(3)
CHARACTER*N STRING
```

DESCRIPTION

DoAnnoText creates an annotation text primitive object that defines a string of text. This primitive object is rendered in a plane in frame space that is parallel to the XY-plane of the display space. Because annotation text has no geometric size, it is unaffected by geometric transformations, projections, lighting and shading.

The textfont can be scaled using *DoTextHeight* <*DOTH*>. Note that the scale factor is specified in frame coordinates.

The parameter *position* determines a three-dimensional point in frame coordinates to specify the text plane for the annotation text. First, *position* is transformed into world coordinates using the current transformation matrix attribute. Then it is mapped into frame space using the current camera projection matrix attribute. This point is the reference point for determining a plane parallel to the XY-plane of device space. This plane becomes the text plane for the annotation text. The reference point, the plane, the *DoTextAlign* <*DOTA*>, *DoTextPath* <*DOTPA*>, and *DoTextUpVector* <*DOTUV*> attributes together determine the orientation of the text string. The parameter *string* is the actual text.

FORTRAN SPECIFIC

In Fortran, *STRING* is a string of *N* characters.

SEE ALSO

DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoBackfaceCullSwitch – Create a primitive attribute object for enabling/disabling backface culling

SYNOPSIS

C:

```
DtObject DoBackfaceCullSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOBFCS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoBackfaceCullSwitch creates a backface cull switch primitive attribute object. The parameter *switchvalue* specifies whether a primitive object will ignore its backward facing surfaces (i.e., those facing away from the viewer) as an efficiency measure when rendering itself. The possible values for *switchvalue* are:

DcOn <DCON>
Backward facing surfaces will be ignored.

DcOff <DCOFF>
Backward facing surfaces will not be ignored.

The geometric normal, either specified explicitly or determined implicitly using the right-hand rule, is used to determine whether a surface is backward facing. For a primitive object to be backface culled, the object must be backface cullable and the backface cull switch must be enabled. In general, backface culling should be enabled for any collection of primitive objects consisting of closed surfaces.

DEFAULTS

The default value for *DoBackfaceCullSwitch* is *DcOn <DCON>*.

SEE ALSO

DoBackfaceCullable(3D), DoHiddenSurfSwitch(3D)

NAME

DoBackfaceCullable – Create a primitive attribute object defining backface cullability

SYNOPSIS

C:

```
DtObject DoBackfaceCullable(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOBFC(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoBackfaceCullable creates a backface cullability primitive attribute object. Backface cullability is a surface property of a primitive object. The *switchvalue* parameter specifies whether a primitive object will render more efficiently if its backward facing surfaces are ignored before shading and rendering takes place. The actual backface culling cannot take place until the *DoBackfaceCullSwitch* <*DOBFCS*> is enabled.

DEFAULTS

The default value for *DoBackfaceCullable* is *DcOff* <*DCOFF*>.

SEE ALSO

DoBackfaceCullSwitch(3D), DoHiddenSurfSwitch(3D)

NAME

DoBoundingVol– Create a bounding volume object

SYNOPSIS

C:

```
DtObject DoBoundingVol(volume, alternate_object)
DtVolume *volume;
DtObject alternate_object;
```

Fortran:

```
INTEGER*4 DOBV(VOLUME, ALTOBJ)
REAL*8 VOLUME(6)
INTEGER*4 ALTOBJ
```

DESCRIPTION

DoBoundingVol returns a bounding volume object. A bounding volume defines a box in modeling coordinates that is assumed to completely enclose all objects below it in the scene data base, i.e., all child branches rooted below that point in the display tree. When a bounding volume object is executed, the given *volume* is projected onto the device. If there is no intersection with the device or if the diagonal of the smallest right rectangle that can enclose this projection in X and Y is smaller than the *DoMinBoundingVolExt <DOMBVE>*, then the rest of the current group is not executed. However, if the bounding volume is visible but under the minimum size, then the optional *alternate_object* is executed.

If there is no alternate object, use a null object pointer, *DcNullObject <DCNULL>*. Then nothing will be drawn if the bounded objects cannot cover the minimum area on the device.

Alternate objects are simpler versions of the objects being skipped to avoid a lot of complex rendering that would contribute little to the final image. One may want to have more than one alternate object; i.e., execute object2 if object1 is too small or execute object3 if object2 is too small, and so on. In this case, the alternate object will be a group object that contains a new shorter minimum bounding extention and a bounding volume containing the even simpler alternate object.

The *volume* parameter is a user-supplied three-dimensional space specifying the bounding volume. *DsCompBoundingVol <DSCBV>* can be used to compute a volume that contains a user-supplied object.

If the current bounding volume switch *DoBoundingVolSwitch <DOBVS>* is *DcOff <DCOFF>*, then bounding volume calculations will be skipped.

SEE ALSO

DoBoundingVolSwitch(3D), DoMinBoundingVolExt(3D), DsCompBoundingVol(3D)

NAME

DoBoundingVolSwitch – Create a bounding volume switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoBoundingVolSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOBVS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoBoundingVolSwitch returns a bounding volume switch primitive attribute object. The bounding volume switch takes one of the following values:

DcOn <DCON>
Enable use of bounding volumes.

DcOff <DCOFF>
Disable use of bounding volumes and ignore all bounding volumes encountered.

DEFAULTS

The default *DoBoundingVolSwitch* is *DcOn <DCON>*.

SEE ALSO

DoBoundingVol(3D), DoMinBoundingVolExt(3D), DsCompBoundingVol(3D)

NAME

DoCallback – Create a callback object

SYNOPSIS

C:

```
DtObject DoCallback(function,dataobject)
DtPtr function;
DtObject dataobject;
```

Fortran:

```
INTEGER*4 DOCB(FUNCT,DATOBJ)
EXTERNAL INTEGER*4 FUNCT
INTEGER*4 DATOBJ
```

DESCRIPTION

DoCallback creates a callback object. The *function* parameter is a pointer to a user-defined function. The *dataobject* parameter is the handle of an object containing the associated run-time data for the user-defined function. This parameter is created with either *DoDataPtr* <*DODP*> or *DoDataVal* <*DODV*>. *DoCallback* calls *function* with the run-time data of *dataobject*.

The format for the user-defined function in C is:

```
user_fcn(data)
DtPtr data;       /* if DoDataPtr was used */
Dt32bits data;    /* if DoDataVal was used */
```

The format for the user-defined function in Fortran is:

```
FCN(DATA)
INTEGER*4 DATA
```

ERRORS

DoCallback will fail if *dataobject* is not a *DataVal*, *DataPtr* or *DcNullObject* <*DCNULL*>.

[WARNING - invalid dataobject handle]

SEE ALSO

DdInqPickCallback(3D), DdSetPickCallback(3D), DoDataPtr(3D), DoDataVal(3D), DsExecuteObj(3D), DsExecutionAbort(3D), DsExecutionReturn(3D)

NAME

DoCamera – Create a camera studio object

SYNOPSIS

C:

DtObject DoCamera()

Fortran:

INTEGER*4 DOCM()

DESCRIPTION

DoCamera creates a camera object. A camera is a studio object. It obtains properties only through attribute inheritance and then only during execution of definition objects.

Cameras obtain their position and orientation through inheritance of the current transformation matrix attribute values. The inverse of the transformation matrix attribute becomes the camera viewing transformation associated with the active camera. *DoLookAtFrom <DOLAF>* is a common mechanism for setting the transformation matrix prior to creating a camera.

The camera object itself is not visible; only its effects are. One may have an arbitrary number of cameras per view, but only one camera may be active at a given time (see *DvSetActiveCamera*). The active camera is the camera used to render the image in the view.

DEFAULTS

The default camera model is a perspective camera with a field of view of 90 degrees, and the *hither* and *yon* planes at -.01 and -1.0, respectively. The camera is positioned at the origin, pointing down the Z-axis in the negative direction. Up is in the direction of the positive Y-axis.

SEE ALSO

DoCameraMatrix(3D), DoLookAtFrom(3D), DoPerspective(3D), DoParallel(3D), DvInqActiveCamera(3D), DvSetActiveCamera(3D)

NAME

DoCameraMatrix – Create a camera matrix studio attribute object.

SYNOPSIS

C:

```
DtObject DoCameraMatrix(matrix)
DtMatrix4x4 matrix;
```

Fortran:

```
INTEGER*4 DOCMX(MATRIX)
INTEGER*4 MATRIX
```

DESCRIPTION

DoCameraMatrix creates a camera matrix studio attribute object. When *DoCameraMatrix* is executed in a definition group, its parameter *matrix* replaces the camera projection matrix attribute used for all subsequently executed camera objects.

The argument *matrix* is an arbitrary 4x4 matrix; singular matrices will cause fatal errors. *DoCameraMatrix* is intended for advanced users needing special effects. Most useful camera matrices are obtainable using by *DoProjection <DOPRJ>*, *DoParallel <DOPAR>*, or *DoPerspective <DOPER>*.

ERRORS

DoCameraMatrix will fail if the matrix is singular.

[WARNING - non-invertible matrix]

SEE ALSO

DoParallel(3D), DoPerspective(3D), DoProjection(3D)

NAME

DoClipSwitch – Create a primitive attribute object for enabling/disabling model clipping

SYNOPSIS

C:

```
DtObject DoClipSwitch(switchvalue)
DtSwitch switchvalue ;
```

Fortran:

```
INTEGER*4 DOCS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoClipSwitch creates a model clipping switch primitive attribute object. Model clipping is used to cut away parts of a scene so that objects that are normally obscured by the clipped objects can now be seen. The *switchvalue* specifies whether or not subsequently executed primitive objects will be clipped against the current model clipping volume. The values for *switchvalue* are:

DcOff <DCOFF>
: Ignore modeling clipping volumes.

DcOn <DCON>
: Clip all subsequent primitive objects against the current model clipping volume.

DEFAULTS

The default *DoClipSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoClipVol(3D)

NAME

DoClipVol – Create a primitive attribute object defining a model clipping volume

SYNOPSIS

C:

```
DtObject DoClipVol(operator, numhalfspaces, halfspaces)
DtClipOperator operator;
DtInt numhalfspaces;
DtHalfSpace halfspaces[];
```

Fortran:

```
INTEGER*4 DOCV(OPRATR, NHLFSP, HLFSPS)
INTEGER*4 OPRATR
INTEGER*4 NHLFSP
REAL*8 HLFSPS(6*NHLFSP)
```

DESCRIPTION

DoClipVol creates a model clipping volume primitive attribute object. This object is used to modify the volume against which subsequently executed primitive objects are clipped when model clipping (or sectioning) is enabled.

The arguments of *DoClipVol* are as follows:

operator
Specifies how the volume defined by *halfspaces* will be combined with the current value of the model clipping volume primitive attribute.

numhalfspaces
Specifies the number of elements in the array *halfspaces*.

halfspaces
Specifies an array of half spaces or sectioning planes.

Each halfspace is defined by a point in model coordinates and a vector that is the normal to the plane defining the boundary of the half space. This normal points in the direction of the acceptance region. The half spaces intersect to define a clipping volume. During display execution, this computed clipping volume is transformed by the current transformation matrix attribute and combined with the current model clipping volume based on the specified operator.

The following table defines the available model clipping constants. T denotes the current value of the model clipping volume attribute, and S denotes the volume defined by the halfspaces.

Fortran Constant	C Constant	Clipping Acceptance Region
DCCALL	*DcClipAll*	everything clipped
DCCAND	*DcClipAnd*	T and S
DCCARV	*DcClipAndReverse*	T and (not S)
DCCNOP	*DcClipNoOp*	T
DCCAIN	*DcClipAndInverted*	(not T) and S
DCCREP	*DcClipReplace*	S
DCCXOR	*DcClipXOr*	T xor S
DCCOR	*DcClipOr*	T or S
DCCNOR	*DcClipNor*	(not T) and (not S)
DCCEQV	*DcClipEqv*	T eqv S [i.e. not (T xor S)]
DCCIVV	*DcClipInvertVolume*	not S
DCCORR	*DcClipOrReverse*	T or (not S)
DCCINV	*DcClipInvert*	not T .

DCCORI	*DcClipOrInverted*	(not T) or S
DCCNAN	*DcClipNAnd*	(not T) or (not S)
DCCNON	*DcClipNone*	nothing clipped

Note that model clipping is considerably more powerful than clipping performed in the viewing operation for the following reasons:

Any number of clipping planes can be specified.

The model clipping planes are oriented arbitrarily with respect to each other and to the coordinate axes.

An arbitrary boolean operator can be used to combine the volume defined by the specified set of half spaces and the current model clipping volume.

DEFAULTS

The default *DoClipVol* is all of world space, i.e., no clipping.

SEE ALSO

DoClipSwitch(3D)

NAME

DoDataPtr – Create a data pointer object

SYNOPSIS

C:

```
DtObject DoDataPtr(dataptr)
DtPtr dataptr;
```

Fortran:

```
INTEGER*4 DODP(DATPTR)
INTEGER*4 DATPTR
```

DESCRIPTION

DoDataPtr creates a data pointer object. The *dataptr* parameter points to a user-defined data structure that is used for passing data to user-defined functions (call-back objects).

SEE ALSO

DdSetPickCallBack(3D), DoCallback(3D), DoDataVal(3D)

NAME

DoDataVal – Create a data value object

SYNOPSIS

C:

```
DtObject DoDataVal(dataval)
Dt32Bits dataval;
```

Fortran:

```
INTEGER*4 DODV(DATVAL)
INTEGER*4 DATVAL
```

DESCRIPTION

DoDataVal creates a data value object. The *dataval* parameter contains a user-defined 32-bit data structure that is used to pass a small user-defined data structure to user-defined functions (callback objects) when these functions are called.

SEE ALSO

DdSetPickCallBack(3D), DoCallback(3D), DoDataPtr(3D)

NAME

DoDepthCue – Create a primitive attribute object defining depth cueing

SYNOPSIS

C:

```
DtObject DoDepthCue(zfront, zback, sfront, sback, colormodel, color)
DtReal zfront, zback, sfront, sback;
DtColorModel colormodel;
DtReal color[ ];
```

Fortran:

```
INTEGER*4 DODC(ZFRONT, ZBACK, SFRONT, SBACK, COLMOD, COLOR)
REAL*8 ZFRONT, ZBACK, SFRONT, SBACK
INTEGER*4 COLMOD
REAL*8 COLOR(*)
```

DESCRIPTION

DoDepthCue creates a depth cue primitive attribute object. The object is used to specify how primitive objects will be depth cued when the depth cue switch primitive attribute (set by *DoDepthCueSwitch* <*DODCS*>) is enabled. Note that depth cueing is on an object-per-object basis.

When the depth cue switch is enabled, a user-provided color is combined with that of any primitive object proportional to each part's distance from the viewer (i.e., Z-values in frame coordinates). If the depth cue switch is disabled, then the primitive object's color is unaffected.

The *zfront* and *zback* values define two planes: Z-values in frame coordinates of two planes parallel to the XY plane. The *sfront* and *sback* values are numbers between 0 and 1 specifying the portion between the primitive's color and user-supplied depth cue color *color* to be used at each plane. The values for Z-values between *sfront* and *sback* are linearly interpolated. Note that the required parameter *colormodel* establishes the color model of the *color* parameter.

The displayed color DC at some depth Z in frame coordinates is related to the primitive's color C, the depthcue color *color*, the front and back Z-planes *zfront* and *zback* (with their associated values *sfront* and *sback*) as follows:

If Z is in front of *zfront*, then

DC = *sfront**C+(1-*sfront*)**color*

If Z is behind *zback*, then

DC = *sback**C+(1-*sback*)**color*

If Z is between *zfront* and *zback* and if

r = *sback*+(((Z-*zback*)*(*sfront*-*sback*))/(*zfront*-*zback*))

then

DC = r*C+(1-r)**color*.

DEFAULTS

The default settings of *DoDepthCue* are (1.0, 0.0, 1.0, 0.0, *DcRGB*, (0.0, 0.0, 0.0)).

SEE ALSO

DoDepthCueSwitch(3D)

NAME

DoDepthCueSwitch – Create a primitive attribute object for enabling/disabling depth cueing

SYNOPSIS

C:

```
DtObject DoDepthCueSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DODCS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoDepthCueSwitch creates a depth cue switch primitive attribute object. The *switchvalue* parameter is used to specify whether or not subsequent primitive objects will be depth cued. When the depth cue switch is enabled, a user-provided color is combined with that of any primitive object proportional to each part's distance from the viewer (i.e., Z-values in frame coordinates). If the depth cue switch is disabled, then the primitive object's color is unaffected.

DEFAULTS

The default for *DoDepthCueSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoDepthCue(3D)

NAME

DoDevice – Open a device

SYNOPSIS

C:

```
DtObject DoDevice(devicetype, argstring)
DtPtr device_type;
DtPtr argstring;
```

Fortran:

```
INTEGER*4 DOD(DEVTYP, LDEV, ARGSTR, LARG)
INTEGER*4 LDEV, LARG
CHARACTER*LDEV DEVTYP
CHARACTER*LARG ARGSTR
```

DESCRIPTION

DoDevice opens and initializes a Doré device of the indicated type. A device is an output mechanism used to display a frame. *DoDevice* returns an object handle to be used for further references to the device.

devicetype is a null terminated string that specifies the type of device to create.

argstring is a null terminated string with embedded device *options*. Options specify optional permanent initialization values for the device. Options not specified will default to reasonable values. An option takes one of the following forms:

-optionflag *optionvalue*
for options requiring values.

-optionflag
for boolean flags where the presence of "-optionflag" means *true* and the absence of "-optionflag" means *false*.

Numeric option values must be specified by their decimal ASCII representations. Multiple options within *argstring* must be separated by at least one blank, tab, or comma, though all legal options may occur in any order within *argstring*. If the same option is specified more than once in *argstring*, then only the last value for that option is used. The legal options in *argstring* depend upon the *devicetype* parameter.

The following two examples list the allowable device options for two different devices. The first example lists the options for a generic raster Doré device, and the second example lists the options for a Stardent 1500/3000 X-based Doré device. For information on the devices available on your Doré configuration, see *Appendix E: Doré Configurations* in the *Doré Reference Guide*. For information on particular devices, see *Appendix F: Doré Device Drivers* in the *Doré Reference Guide*. In some cases, pertinent information may be found in the release notes accompanying the Doré implementation.

* "rasterfile" - a simple file format for full-color pixel data output.

"-filename *name*"
where *name* is a string that specifies the name of the file into which the image should be stored.

"-width *pixwidth*"
where *pixwidth* is the width in pixels of the image to be saved. The default width is 512 pixels if this parameter is not specified.

"-height *pixheight*"
where *pixheight* is the height in pixels of the image to be saved. The default height is 512 pixels if this parameter is not specified.

* "stardentx11" - a dynamic x11 window device.

"-singlebuffered"
requests a single buffered window.

"-geometry *geomstring*"
requests a window with a particular position and size. The format for *geomstring* is "WxH+X+Y" or "WxH" (where W, H, X, and Y are the integer values for width, height, and the X,Y position of the upper left corner of the window). If the "WxH" format is used, X and Y are assumed to be zero.

"-visualtype *vtype*"
requests a window with of the given visual type (where *vtype* is *DcDirectColor, DcPseudoColor,* etc.).

"-window *xwindow*"
passes to the handle of a window that was opened by the application and into which Doré is to draw. When this option is included, options of the previous three types will be ignored.

"-display *xdisplay*"
If a window option was specified, that window on the display *xdisplay* is used. Otherwise, Doré opens a window on *xdisplay*.

"-stereo"
requests a stereo window.

After a call to *DoDevice* and until the device object is deleted, the device handle may be used by subsequent device functions to alter the characteristics of the actual device (such as causing it to display). Doré devices use right-handed 3–D floating point coordinate systems with the origin in the back, lower, left corner of the actual device.

FORTRAN SPECIFIC

DEVTYP is a string *LDEV* bytes long containing the type of device to create.

ARGSTR is a string *LARG* bytes long containing device options.

ERRORS

DoDevice will fail if the specified device is unavailable or an illegal device type or option is specified.

[SEVERE - unable to allocate device]

DoDevice will fail if enough memory for the device cannot be allocated

[SEVERE - unable to allocate memory]

SEE ALSO

DdUpdate(3D), DoFrame(3D)

NAME

DoDiffuseColor – Create a diffuse color primitive attribute object

SYNOPSIS

C:

```
DtObject DoDiffuseColor(colormodel, color)
DtColorModel colormodel;
DtReal color[ ];
```

Fortran:

```
INTEGER*4 DODIFC(COLMOD, COLOR)
INTEGER*4 COLMOD
REAL*8 COLOR(3)
```

DESCRIPTION

DoDiffuseColor creates an diffuse color primitive attribute using the color model specified by *colormodel*. The *color* parameter specifies the diffuse response of a surface to light.

Diffuse radiation is the response of a surface to incident (i.e., non-ambient) light where the emitted light is scattered equally in all directions. The diffuse color of a surface is often thought of as the base color. The diffuse color is also used for determining the color of the ambient component of a surface.

The diffuse color is one of three aspects of the diffuse component of a surface. The other two components are *DoDiffuseIntens* <*DODIFI*> and *DoDiffuseSwitch* <*DODIFS*>.

DEFAULTS

The default *DoDiffuseColor* is (*DcRGB*, (1.0, 1.0, 1.0)).

SEE ALSO

DoDiffuseIntens(3D), DoDiffuseSwitch(3D)

NAME

DoDiffuseIntens – Create a diffuse intensity primitive attribute object

SYNOPSIS

C:

```
DtObject DoDiffuseIntens(intensity)
DtReal intensity;
```

Fortran:

```
INTEGER*4 DODIFI(INTENS)
REAL*8 INTENS
```

DESCRIPTION

DoDiffuseIntens creates a diffuse intensity primitive attribute. The *intensity* parameter specifies the intensity of a surface's diffuse response to light from non-ambient light sources in the environment. The diffuse intensity normally ranges from 0.0 to 1.0, signifying the contribution of diffuse color to the overall shade of the object's surface.

The diffuse intensity is one of three aspects of the diffuse component of a surface. The other two aspects are the *DoDiffuseColor* <*DODIFC*> and the *DoDiffuseSwitch* <*DODIFS*>.

DEFAULTS

The default value for *DoDiffuseIntens* is 1.0.

SEE ALSO

DoDiffuseColor(3D), DoDiffuseSwitch(3D)

NAME

DoDiffuseSwitch – Create a diffuse switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoDiffuseSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DODIFS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoDiffuseSwitch creates a diffuse switch primitive attribute. The *switchvalue* parameter is used to specify whether or not the surfaces of subsequent primitive objects have a diffuse component in their responses to light from non-ambient light sources in the environment.

The diffuse switch determines whether diffuse attributes will be used. If the *switchvalue* is *DcOff* <*DCOFF*>, the primitive object will render itself without diffuse shading. If the *switchvalue* is *DcOn* <*DCON*>, the primitive object will render itself using the diffuse attributes whenever possible.

The diffuse switch is one of three aspects of the diffuse component of a surface. The other two components are the *DoDiffuseColor* <*DODIFC*> and the *DoDiffuseIntens* <*DODIFI*>.

DEFAULTS

The default for *DoDiffuseSwitch* is *DcOn* <*DCON*>.

SEE ALSO

DoDiffuseColor(3D), DoDiffuseIntens(3D)

NAME

DoExecSet – Create an object that modifies the executability of objects

SYNOPSIS

C:

```
DtObject DoExecSet(n, list, setop)
DtInt    n;
DtInt    list[];
DtSetOperation setop;
```

Fortran:

```
INTEGER*4 DOES(N, LIST, SETOP)
INTEGER*4 N
INTEGER*4 LIST(N)
INTEGER*4 SETOP
```

DESCRIPTION

DoExecSet creates an organizational object that modifies the executability set. The executability set is a set of object types that are currently eligible to be executed as definition or display objects. By default, all object types are included in this set which allows them to execute the current method.

The parameter *list* is a list of object types. *n* is the number of elements in the list. Each element of the list refers to a member of the executability set (an object type) to be affected. The argument *setop* specifies how the status of those members are to be affected. Possible choices for *setop* are:

DcSetAdd <DCSADD>
: Adds the object types listed in *list* from the current executability set.

DcSetDelete <DCSDEL>
: Removes the object types listed in *list* from the current executability set.

DcSetInvert <DCSINV>
: Causes all object types referred to in *list* to take on the opposite of their current status. In other words, if the given object types are not members of the current executability set, they will added; if they currently are included, they will be removed.

DcSetReplace <DCSREP>
: Replaces the current executability set with one that includes only the object types given in *list*.

When an object type is non-executable, the effect is the same as if no objects of that type existed below that point in the tree until that object type is made executable again. This is useful for making a temporary global change to an entire subtree.

For instance, imagine a complex scene containing hundreds of objects that are defined with many different representation types. To render either the entire scene or some portion of it all in the same representation type would require extensive editing of the scene. The same could be accomplished using *DoExecSet* by adding just two new group elements at the beginning of the group to be affected: one to set the desired representation type followed by one to remove represention type from the current executability set. Other object types subject to this form of manipulation include any of the appearance attributes. In fact, some very bizarre effects can result from disabling other group element types like geometric attributes.

Note that this is a useful, flexible, and powerful function but capable of disastrous effects if not used carefully. It is highly recommended that this be used only to *temporarily* disable executability of one or more attribute objects for a single subtree. It is

further advised that *DcSetDelete* <*DCSDEL*> be the only *setop* used as above; the other set operations exist mainly for the sake of consistency.

ERRORS

DoExecSet will fail if the *setop* value is invalid.

[FATAL - invalid set operation]

DEFAULTS

The default *DoExecSet* includes all currently defined objects that can be placed in the database.

SEE ALSO

DoNameSet(3D), DoFilter(3D)

NAME

DoFileRaster – Create a fileraster object

SYNOPSIS

C:

```
DtObject DoFileRaster(filename, specialstring)
DtPtr filename;
DtPtr specialstring;
```

Fortran:

```
INTEGER*4 DOFRS(FNAME, FLEN, SPCSTR, SLEN)
INTEGER*4 FLEN
CHARACTER*FLEN FNAME
INTEGER*4 SLEN
CHARACTER*SLEN SPCSTR
```

DESCRIPTION

DoFileRaster creates a fileraster object. Fileraster objects can, for example, be used as texture maps or as backgrounds for views.

The parameter *filename* is the name of a file containing raster data. If the file is a standard Doré raster file then *specialstring* should be set to *DcNullPtr* <*DCNULL*>. Some renderers may require special file formats for raster data, and the character string *specialstring* is used to specify the format of the file.

FORTRAN SPECIFIC

The parameter *FNAME* is the name of a file *FLEN* characters long containing raster data. The parameter *SPCSTR* is an optional format specification which is *SLEN* characters in length.

SEE ALSO

DoFileRasterRead(3D), DoRaster(3D)

NAME

DoFilter – Create a filter modifying object

SYNOPSIS

C:

```
DtObject DoFilter(filter, n, members, setop)
DtFilter filter;
DtInt    n;
DtInt    members[];
DtSetOperation setop;
```

Fortran:

```
INTEGER*4 DOFL(FILTER, N, MEMBER, SETOP)
INTEGER*4 FILTER
INTEGER*4 N
INTEGER*4 MEMBER(N)
INTEGER*4 SETOP
```

DESCRIPTION

DoFilter creates an organizational object for modifying filter attributes. Filter attributes, in conjunction with nameset attributes, provide a high-level control over the invisibility and pickability.

Namesets and filters are specified as enumerated sets of members. Currently Doré supports up to 256 members referred to by the numbers 0 through 255. The meaning of a member is defined by the user. For example, the user could reserve member number 0 to indicate red objects and member number 9 to indicate spheres. Then an object whose name set includes members 0 and 9 indicates that that object is a red sphere while a nameset including only member number 9 indicates a non-red sphere.

There exists an inclusion and an exclusion filter for invisibility and pickability. Each quality is enabled if the intersection of the current nameset with the current inclusion filter for that quality is not empty and if the intersection of the current nameset with the corresponding exclusion filter is empty. These qualities are enabled and disabled with the corresponding switch primitive attributes.

Like namesets, filter attribute values are calculated during execution.

Invisibility means that primitive objects will not be displayed. This applies to picking as well as rendering; i.e., invisible objects cannot be picked. Pickability means that objects are eligible to be picked during picking (see *DdPickObjs*).

The parameter *filter* specifies the current filter to be affected. Possible choices are:

DcInvisibilityInclusion <*DCINVI*>
DcInvisibilityExclusion <*DCINVE*>
DcPickabilityInclusion <*DCPCKI*>
DcPickabilityExclusion <*DCPCKE*>

The parameter *members* is a list of members of the specified filter to be affected. *n* is the number of elements in the list. The parameter *setop* specifies how the status of the members referred to in *members* is to be affected. The valid choices for *setop* are:

DcSetAdd <*DCSADD*>
Adds members listed in *members* to the current specified filter.

DcSetDelete <*DCSDEL*>
Removes members listed in *members* from the current specified filter.

DcSetInvert <*DCSINV*>
Causes all members of a filter listed in *members* to take on the opposite of their current values. In other words, if the given members are not in the the current filter, they will be added; if they are included, they will be removed.

DcSetReplace <*DCSREP*>
Replaces the current value of the specified filter with one that includes only the members listed in *members*.

ERRORS

DoFilter will fail if the *setop* value is invalid.

[FATAL - invalid set operation]

DEFAULTS

All filters default to empty, implying that all primitive objects are not invisible and not pickable.

SEE ALSO

DoInvisSwitch(3D), DoNameSet(3D), DoPickSwitch(3D)

NAME

DoFrame – Create a frame object

SYNOPSIS

C:

DtObject DoFrame()

Fortran:

INTEGER*4 DOFR()

DESCRIPTION

DoFrame creates a new frame organizational object. A frame is an object that describes an image to be displayed on a device. It defines a device independent 3D local coordinate system within which one or more views can be positioned.

A frame can be displayed on 0 or more devices using the function *DdSetFrame <DDSF>*. The frame is mapped onto the largest right rectangular volume that can fit within the device viewport such that the aspect ratio in X and Y is preserved and that the Z extent of the frame is mapped to the entire Z extent of the device. When this mapping is performed, there may be extra white space inside the device viewport. The function *DfSetJust <DFSJ>* is used to position this white space.

DEFAULTS

The frame extends from (0.0, 0.0, 0.0) to (1.0, 1.0, 1.0) with the volume specified by *DfSetBoundary <DFSB>* and queried with *DfInqBoundary <DFQB>*.

SEE ALSO

DdSetFrame(3D), DfInqBoundary(3D), DfInqJust(3D), DfSetBoundary(3D), DfSetJust(3D), DfSetFrameJust(3D), DfUpdate(3D)

NAME

DoGenerateTextureUV – Create a primitive attribute object for enabling/disabling generation of **uv** texture coordinates

SYNOPSIS

C:

```
DtObject DoGenerateTextureUV(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOGTUV(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoGenerateTextureUV creates a primitive attribute object that enables/disables the generation of **uv** texture coordinates for primitives that do not have user-provided vertex information.

For some renderers certain primitive objects, such as spheres, are decomposed into triangles before they are drawn. If a sphere is to be texture mapped, the triangle vertices approximating the sphere must include **uv** and/or **uvw** texture coordinates. On the other hand if the sphere will never be texture mapped there is no need to generate and store that information.

The *switchvalue* parameter is used to specify whether or not subsequent primitives will generate **uv** coordinates for texture mapping when they decompose themselves into triangles. If the *switchvalue* is *DcOff* <*DCOFF*>, the primitive object will not generate any **uv** information. If the *switchvalue* is *DcOn* <*DCON*>, the primitive object will generate **uv** coordinates when possible.

DEFAULTS

The default for *DoGenerateTextureUV* is *DcOff*.

SEE ALSO

DoTextureMapBump(3D), DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapTranspIntens(3D)

NAME

DoGlbRendMaxObjs – Create a studio attribute object that defines the maximum number of objects per spatial subdivision for global ray casting

SYNOPSIS

C:

```
DtObject DoGlbRendMaxObjs(maxobjs)
DtInt maxobjs;
```

Fortran:

```
INTEGER*4 DOGRMO(MAXOBJ)
INTEGER*4 MAXOBJ
```

DESCRIPTION

DoGlbRendMaxObjs creates a studio attribute object that defines the maximum number of objects per spatial subdivision for global ray casting. The parameter *maxobjs* specifies the maximum number of geometric entities allowed per spatial subdivision prior to ray casting.

DoGlbRendMaxObjs affects only renderers that use ray casting. The rendering style is set with *DvSetRendStyle <DVSRS>*.

Ray casting is the process of determining which objects in the scene are in the path of an arbitrary vector (ray) in space. Ray casting is used in the generation of global shading effects like shadows, object-to-object reflections, and transparency.

Many renderers use a spatial-subdivision based ray casting algorithm to facilitate the ray casting process in complex scenes containing many geometric entities. A preprocessing step is performed on the scene to collect information about all geometric entities in the environment and then to perform a regular binary subdivision of all space containing geometric entities. The subdivision creates a three-dimensional grid of boxes; i.e, the first level subdivision creates 8 equal boxes placed 2 wide by 2 high by 2 deep, fully subdivided space to two levels creates 64 boxes, and so on. Each box can contain 0 or more geometric entities with more boxes wherever the density of geometric entities is higher. In Doré, the amount of spatial subdivision performed is controlled by these factors:

The number, density, and distribution of geometric entities in the scene.

The user-defined control, *DoGlbRendMaxObjs*.

The user-defined control, *DoGlbRendMaxSub <DOGRMS>*.

DoGlbRendMaxObjs determines the maximum number of geometric entities that a box may contain without being further subdivided. For example, if the maximum number is set to 2, then boxes containing 2 and fewer entities will not be subdivided further; boxes containing more than 2 will be subdivided. Often a given area of space will always contain more than the maximum number of geometric entities; i.e., if 5 geometric entities intersect at a given point, then the area of space containing that point will never contain less than 5 geometric entities.

DoGlbRendMaxSub <DOGRMS> tells the preprocessor how many levels to subdivide before subdivision of that branch is terminated; i.e., it tells the preprocessor when to stop subdividing regardless of the number of geometric entities per box.

In general, the fewer geometric entities per subdivision, the easier to cast rays through each box.

DEFAULTS

The default value for *DoGlbRendMaxObjs* is 1.

SEE ALSO

DoGlbRendMaxSub(3D), DoGlbRendRayLevel(3D)

NAME

DoGlbRendMaxSub – Create a studio attribute object that defines the maximum number of subdivisions for global ray casting

SYNOPSIS

C:

```
DtObject DoGlbRendMaxSub(maxsub)
DtInt maxsub;
```

Fortran:

```
INTEGER*4 DOGRMS(MAXSUB)
INTEGER*4 MAXSUB
```

DESCRIPTION

DoGlbRendMaxSub creates a studio attribute object that defines the maximum number of branch subdivisions for global ray casting. The parameter *maxsub* specifies the maximum level of adaptive spatial subdivision prior to ray casting. See *DoGlbRendMaxObjs* for a detailed explanation of spatial subdivision.

DEFAULTS

The default value for *DoGlbRendMaxSub* is 10.

SEE ALSO

DoGlbRendMaxObjs(3D), DoGlbRendRayLevel(3D)

NAME

DoGlbRendRayLevel – Create a ray level studio attribute object

SYNOPSIS

C:

```
DtObject DoGlbRendRayLevel(raylevel)
DtInt raylevel;
```

Fortran:

```
INTEGER*4 DOGRRL(RAYLEV)
INTEGER*4 RAYLEV
```

DESCRIPTION

DoGlbRendRayLevel creates a ray level studio attribute object. The parameter *raylevel* specifies the maximum number of times a ray will be allowed to bounce from primitive objects during ray casting. Ray casting is the process of determining which objects in the scene are in the path of an arbitrary vector (ray) in space. Ray casting is used in the generation of global shading effects like shadows, object-to-object reflections, and transparency.

DoGlbRendRayLevel affects only renderers that use ray casting. The rendering style is set with *DvSetRendStyle <DVSRS>*.

DEFAULTS

The default value for *DoGlbRendRayLevel* is 3.

SEE ALSO

DoGlbRendMaxObjs(3D), DoGlbRendMaxSub(3D)

NAME

DoGroup – Create a group organizationalal object

SYNOPSIS

C:

```
DtObject DoGroup(open)
DtFlag open;
```

Fortran:

```
INTEGER*4 DOG(OPEN)
INTEGER*4 OPEN
```

DESCRIPTION

DoGroup creates a new group organizational object. A group is an organizational object that contains other objects called the elements of the group. A group can be thought of as a simple list. The parameter *open* determines whether the group is created open, i.e., whether the group is available for edit. Groups are edited using the following functions: *DgAddObj <DGAO>, DgAddObjToGroup <DGAOG>, DgReplaceObj <DGRO>, DgReplaceObjInGroup <DGROG>, DgDelEleBetweenLabels <DGDEL>, DgDelEle <DGDE>,* and *DgDelEleRange <DGDER>*. An existing group can be opened with *DgOpen <DGO>*

An element pointer is used by many of the editing functions to designate the element(s) to be edited. There is one element pointer per group, and it remembers its last location. Label objects also may be used within the group to identify key objects for some editing functions. (See *DoLabel.*) The application should close a group when it has finished editing the group. (see *DgClose*)

A group network consists of a group and all the groups referenced, either directly or indirectly, by that group. Group networks can be very complex; nothing prevents them from including arbitrary cyclic or recursive references. These recursive references are not supported and should be avoided by the application because they cause unpredictable and often fatal results. (See *DgCheck.*)

Groups can contain other groups, allowing an application to construct hierarchies of groups to match the hierarchical structure of the application model. A single group may be a child group of more than one group; i.e., the same group may be instanced in several places. Each time the group is instanced, it inherits the current attributes wherever it is instanced.

When one group references another group, the referenced group starts with a complete set of the current appearance and geometric attributes of the referencing group. The referenced group may add its own geometric transformations or appearance attributes for its own primitive objects and then pass the new set of attributes on to any groups it references.

There are two kinds of groups, regular groups and inline groups; these differ only in the way they inherit attributes.

In a regular group, a copy of all attributes is saved upon entering the group and restored when execution of the group terminates. This pushing and popping of attributes directs inheritance in one direction only; attributes changed within a referenced group do not affect the attributes of the referencing group.

Inline groups, created with *DoInLineGroup <DOILG>*, are slightly different. They do not push and pop the current attributes upon entry and exit. Changes made to attributes by an inline group have the same effect as if they had been placed directly in the calling group. The inline group thus provides a mechanism for defining a commonly used set of attributes once and then using the set multiple times.

SEE ALSO

DgAddObj(3D), DgAddObjToGroup(3D), DgClose(3D), DgDelEle(3D), DgDelEleBetweenLabels(3D), DgDelEleRange(3D), DgEmpty(3D), DgInqElePtr(3D), DgInqSize(3D), DgInqObjAtPos(3D), DgInqOpen(3D), DgOpen(3D), DgReplaceObj(3D), DgSetElePtr(3D), DgSetElePtrRelLabel(3D), DoInLineGroup(3D), DoLabel(3D)

NAME

DoHiddenSurfSwitch – Create a hidden surface switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoHiddenSurfSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOHSS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoHiddenSurfSwitch creates a hidden surface primitive attribute object. The parameter *switchvalue* specifies whether all portions of subsequent primitive objects will be rendered, including those that are obscured by other primitive objects or by other portions of the same object.

If *switchvalue* is *DcOff* <*DCOFF*>, primitive objects render themselves completely even if they are partially or completely hidden with respect to the viewer by other primitive objects in the scene. Hence, the order in which primitive objects are displayed is critial to the scene organization. If *switchvalue* is *DcOn* <*DCON*>, primitive objects do not render the portions of themselves that are hidden with respect to the viewer by other objects in the scene.

DEFAULTS

The default value for *DoHiddenSurfSwitch* is *DcOn* <*DCON*>.

SEE ALSO

DoBackfaceCullSwitch(3D)

NAME

DoInLineGroup – Create an inline group object

SYNOPSIS

C:

```
DtObject DoInLineGroup(open)
DtFlag open;
```

Fortran:

```
INTEGER*4 DOILG(OPEN)
INTEGER*4 OPEN
```

DESCRIPTION

DoInLineGroup creates an inline group organizational object. A group is an organizational object that contains other objects called the elements of the group. A group can be thought of as a simple list. The parameter *open* determines whether the group is created open, i.e., whether the group is immediately available for object additions. If *DcTrue* <*DCTRUE*> is specified, the group becomes active and available for additions.

There are two kinds of groups, regular groups and inline groups. They differ only in the way they inherit attributes. In a regular group, a copy of all attributes is saved upon entering the group and restored when execution of the group terminates. Inline groups, on the other hand, do not push and pop the current attributes upon their entry and exit. Changes made to attributes by an inline group have the same effect as if they had been placed directly in the calling group. The inline group thus provides a mechanism for defining a commonly used set of attributes once and then using that set multiple times.

SEE ALSO

DgOpen(3D), DoGroup(3D)

NAME

DoInputSlot– Create an input slot organizationalal object

SYNOPSIS

C:

DtObject DoInputSlot()

Fortran:

INTEGER*4 DOIS()

DESCRIPTION

DoInputSlot returns an input slot organizational object. An input slot is an entry point for decimal values. It is the primary interface for connecting input events and values to Doré. Values generated by either internal or asynchronous external events can be input via the function *DsInputValue <DSIV>*, which takes a slot object and a *DtReal <REAL*>* value and triggers all the valuators attached to that slot. A valuator is a callback object that depends upon input slots. See *DsInqValuatorGroup* for information on how to create a valuator.

SEE ALSO

DsInqValuatorGroup(3D), DoCallback(3D)

NAME

DoInterpType – Create an interpolation type primitive attribute object

SYNOPSIS

C:

```
DtObject DoInterpType(type)
DtInterpType type;
```

Fortran:

```
INTEGER*4 DOIT(TYPE)
INTEGER*4 TYPE
```

DESCRIPTION

DoInterpType creates an interpolation type primitive attribute object. The parameter *type* specifies the information interpolated by the object when it renders itself.

The parameter *type* argument can have the following values:

DcConstantShade <*DCCNSH*>
: Average the information from each vertex and generate a single shade to be used on the entire object.

DcVertexShade <*DCVXSH*>
: Render each vertex of the object and shade the entire object as a linearly interpolated shade of the vertex colors.

DcSurfaceShade <*DCSFSH*>
: Interpolate all information from each vertex linearly across the object and then render each point on the object.

DEFAULTS

The default value for *DoInterpType* is *DcConstantShade* <*DCCNSH*>.

SEE ALSO

DoRepType(3D)

NAME

DoInvisSwitch – Create an invisibility switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoInvisSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOINVS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoInvisSwitch creates an invisibility attribute. The parameter *switchvalue* determines whether subsequent primitive objects will be rendered. The parameter takes one of the following values:

DcOff <DCOFF>
Render subsequent primitive objects normally.

DcOn <DCON>
Do not render subsequent primitive objects.

DoInvisSwitch provides direct control of the invisibility primitive attribute at the lowest level. Higher level control is provided through namesets and filters.

DEFAULTS

The default value for *DoInvisSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoFilter(3D), DoNameSet(3D)

NAME

DoLabel – Create a label organizational object

SYNOPSIS

C:

```
DtObject DoLabel(label)
DtInt label;
```

Fortran:

```
INTEGER*4 DOLL(LABEL)
INTEGER*4 LABEL
```

DESCRIPTION

DoLabel creates a label organizational object. The parameter *label* marks a special location in the group. They can be placed anywhere within the group. There may be more than one location with the same label; however, functions using labels search for the first occurrence of a given label starting at the position of the current group element pointer. Labels have no effect during group execution.

SEE ALSO

DgDelEleBetweenLabels(3D), DgInqElePtr(3D), DgSetElePtrRelLabel(3D)

NAME

DoLight – Create a light studio object

SYNOPSIS

C:

DtObject DoLight()

Fortran:

INTEGER*4 DOLT()

DESCRIPTION

DoLight creates a light object. A light is a studio object. It obtains properties only through attribute inheritance and then only during execution of definition objects. Lights obtain their position and orientation through inheritance of the current transformation matrix attribute.

All lights have intensity and color and are of a specific type. The intensity and color are determined by *DoLightIntens <DOLI>* and *DoLightColor <DOLC>*. The light type is determined by *DoLightType <DOLNT>*. Depending on the type, some lights also have one or more of the following: position, direction, attenuation, spread angles, and spread exponent.

Light objects are not visible; only their effects are. One may have an arbitrary number of lights per view. Lights can be turned on and off per primitive object in the scene (see *DoLightSwitch*).

DEFAULTS

The default light model is an infinite light source of intensity 1.0 located at the origin and pointing down the Z-axis in the negative direction.

SEE ALSO

DoLightColor(3D), DoLightIntens(3D), DoLightType(3D)

NAME

DoLightAttenuation – Create a light attribute object defining light intensity falloff with distance

SYNOPSIS

C:

```
DtObject DoLightAttenuation(c1, c2)
DtReal c1;
DtReal c2;
```

Fortran:

```
INTEGER*4 DOLTA(C1, C2)
REAL*8 C1
REAL*8 C2
```

DESCRIPTION

DoLightAttenuation creates a light studio attribute object that defines a multiplicative factor modifying the light intensity as a function of the distance from the light source. The parameters *c1* and *c2* are constants used to calculate this factor:

attn = 1. / (*c1* + *c2**distance)

The *DoLightAttenuation* attribute defines the light attenuation for lights of type *DcLightPointAttn* <*DCLTPA*> and *DcLightSpotAttn* <*DCLTSA*>.

ERRORS

DEFAULTS

The default *DoLightAttenuation* is (1., 1.).

SEE ALSO

DoLightSpreadAngles(3D), DoLightSpreadExp(3D) DoLightType(3D)

NAME

DoLightColor – Create a light color studio attribute object

SYNOPSIS

C:

```
DtObject DoLightColor(colormodel, color)
DtColorModel colormodel;
DtReal color[ ];
```

Fortran:

```
INTEGER*4 DOLC(COLMOD, COLOR)
INTEGER*4 COLMOD
REAL*8 COLOR()
```

DESCRIPTION

DoLightColor creates a light color studio attribute object. The parameter *colormodel* specifies the kind of color used. The parameter *color* specifies the color, in the designated color model, of subsequent light objects.

The light color attribute controls only the chromatic composition of the light emitted by a light source; the intensity of the light source is determined by the light intensity attribute.

DEFAULTS

The default *DoLightColor* is (*DcRGB*, (1.0, 1.0, 1.0)).

SEE ALSO

DoLight(3D), DoLightIntens(3D)

NAME

DoLightIntens – Create a light intensity studio attribute object

SYNOPSIS

C:

```
DtObject DoLightIntens(intensity)
DtReal intensity;
```

Fortran:

```
INTEGER*4 DOLI(INTENS)
REAL*8 INTENS
```

DESCRIPTION

DoLightIntensity creates a light intensity studio attribute object. The parameter *intensity* specifies the overall intensity of subsequent light objects. It may be any non-negative real number; however, normal light intensities range between 0.0 and 1.0.

ERRORS

DoLightIntens issue a warning if *intensity* is negative; the intensity will be set to 0.0.

[WARNING - negative light intensity]

DEFAULTS

The default *DoLightIntens* is 1.0.

SEE ALSO

DoLight(3D), DoLightColor(3D)

NAME

DoLightSpreadAngles – Create a light attribute object defining the width of the light beam

SYNOPSIS

C:

```
DtObject DoLightSpreadAngles(total_angle, delta_angle)
DtReal total_angle;
DtReal delta_angle;
```

Fortran:

```
INTEGER*4 DOLTSA(TANGLE, DANGLE)
REAL*8 TANGLE
REAL*8 DANGLE
```

DESCRIPTION

DoLightSpreadAngles creates a light studio attribute object that defines a multiplicative factor modifying the light intensity as a function of the angle from the light direction. The *DoLightSpreadAngles* attribute defines the light cone for lights of type *DcLightSpot <DCLTSP>* or *DcLightSpotAttn <DCLTSA>*. *total_angle* is the angle (in radians), measured from the light direction, in which the light intensity is non-zero. For any point *p*, a line can be drawn from *p* to the light position. If the angle this line makes with the light direction vector is between zero and (*total_angle - delta_angle*) the multiplicative factor is one and zero for angles greater that *total_angle*. For values between (*total_angle - delta_angle*) and *total_angle* the multiplicative factor is a ramp from 1 to 0.

ERRORS

DoLightSpreadAngles will fail if *delta_angle* is greater than *total_angle* or less then zero, or if *total_angle* is less than zero.

[WARNING - value out of range]

DEFAULTS

The default value for *DoLightSpreadAngles* is (pi/4, 0).

SEE ALSO

DoLightSpreadExp(3D), DoLightAttenuation(3D)

NAME

DoLightSpreadExp – Create a light attribute object defining light intensity falloff with angle

SYNOPSIS

C:

```
DtObject DoLightSpreadExp(exponent)
DtReal exponent;
```

Fortran:

```
INTEGER*4 DOLSE(EXPN)
REAL*8 EXPN
```

DESCRIPTION

DoLightSpreadExp creates a light studio attribute object that defines a multiplicative factor modifying the light intensity as a function of the angle from the light direction. For any point *p*, a line can be drawn from *p* to the light position. The cosine of the angle between this line and the light direction vector raised to the power *exponent* is used as a multiplicative factor modifying the light intensity at the point *p*.

ERRORS

DoLightSpreadExp will fail if *exponent* is less than zero.

[WARNING - value out of range]

DEFAULTS

The default value for *DoLightSpreadExp* is 0.

SEE ALSO

DoLightSpreadAngles(3D), DoLightAttenuation(3D), DoLightType(3D)

NAME

DoLightSwitch – Create a primitive attribute object enabling/disabling illumination from a light

SYNOPSIS

C:

```
DtObject DoLightSwitch(light, switchvalue)
DtObject light;
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOLTS(LIGHT, SWVAL)
INTEGER*4 LIGHT
INTEGER*4 SWVAL
```

DESCRIPTION

DoLightSwitch creates a light switch primitive attribute. The parameter *light* is a handle to a light object that has been created with *DoLight* <DOLT>. The *switchvalue* parameter is used to specify whether or not the surfaces of subsequent primitive objects are affected by the light *light*. If the *switchvalue* is *DcOff*, the primitive object will ignore the specified light when rendering itself, regardless of the type of shading. If the *switchvalue* is *DcOn*, the primitive object will render itself using illumination from the specified light whenever possible.

ERRORS

DoLightSwitch will fail if *light* is not a valid light object.

[WARNING - invalid or deleted object]

DEFAULTS

By default all primitive objects are affected by all defined lights.

SEE ALSO

DoLight(3D)

NAME

DoLightType – Create a light type studio attribute object

SYNOPSIS

C:

```
DtObject DoLightType(type)
DtObject type;
```

Fortran:

```
INTEGER*4 DOLTT(TYPE)
INTEGER*4 TYPE
```

DESCRIPTION

DoLightType creates a light type studio attribute object. The parameter *type* specifies the lighting model for subsequent lights. Six light models are currently provided:

DcLightAmbient <DCLTAM>
Light from the light source is equally distributed throughout the scene. The total ambient light in the scene is the combination of all ambient lights in the scene.

DcLightInfinite <DCLTIN>
The light source is at infinity. All rays of light entering the scene will appear to be parallel. The vector (0.,0.,-1.) is transformed by the current transformation matrix to determine the light direction.

DcLightPoint <DCLTPT>
The light source is a point source that radiates light outward uniformly in all directions from the light position. The point (0.,0.,0.) is transformed by the current transformation matrix to determine the light position.

DcLightPointAttn <DCLTPA>
The light source is just like *DcLightPoint <DCLTPT>* but with the additional attribute *DoLightAttenuation* controlling the falloff of the light intensity as a function of the distance from the light source.

DcLightSpot <DCLTSP>
The light source is a directed point source that radiates light outward from the light position and whose light intensity decreases as a function of the angle from the light direction. The point (0.,0.,0.) is transformed by the current transformation matrix to determine the light position. The vector (0.,0.,-1.) is transformed by the current transformation matrix to determine the light direction. The light attributes *DoLightSpreadAngles* and *DoLightSpreadExp* control the width and falloff of the light intensity as a function of the angle from the light direction.

DcLightSpotAttn <DCLTSA>
The light source is just like *DcLightSpot <DCLTSP>* but with the additional attribute *DoLightAttenuation* controlling the falloff of the light intensity as a function of the distance from the light source.

DEFAULTS

The default *DoLightType* is *DcLightInfinite <DCLTIN>*.

SEE ALSO

DoLight(3D), DoLightSpreadAngles(3D), DoLightSpreadExp(3D), DoLightAttenuation(3D)

NAME

DoLineList – Create a line list primitive object

SYNOPSIS

C:

```
DtObject DoLineList(colormodel, vertextype, linecount, vertices)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt linecount;
DtReal vertices[];
```

Fortran:

```
INTEGER*4 DOLINL(COLMOD, VTXTYP, LINCNT, VTXS)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 LINCNT
REAL*8 VTXS(*)
```

DESCRIPTION

DoLineList creates a primitive object that defines a list of independent lines. The parameter *colormodel* specifies the color model to be used when defining vertex colors (if any). The parameter *vertextype* specifies the exact nature of the vertex: whether the vertex contains color and/or normal information for shading purposes. See the appendix on vertex types for more information. The parameter *linecount* contains the number of lines in the list. The parameter *vertices* is an array of three dimensional vertices containing data for 2**linecount* vertices.

When rendered, the *DoLineList* primitive object uses the *DoLineWidth* <*DOLW*> and *DoLineType* <*DOLNT*> primitive attributes to determine the current line width and style.

SEE ALSO

DoLineType(3D), DoLineWidth(3D), VertexTypes(3D)

NAME

DoLineType – Create a line type primitive attribute object

SYNOPSIS

C:

```
DtObject DoLineType(type)
DtLineType type;
```

Fortran:

```
INTEGER*4 DOLNT(TYPE)
INTEGER*4 TYPE
```

DESCRIPTION

DoLineType creates a line type primitive attribute object. This object causes all subsequent primitive objects defined as lines, curves, or combinations of the two to be drawn with the appropriate line type. Possible values for *type* are:

DcLineTypeSolid <DCLTS>
Solid

DcLineTypeDash <DCLTDS>
Dashed

DcLineTypeDot <DCLTDT>
Dotted

DcLineTypeDotDash <DCLTDD>
Dot and dash repeated

DEFAULTS

The default *DoLineType* is *DcLineTypeSolid <DCLTS>*.

SEE ALSO

DoLineList(3D), DoLineWidth(3D), DoPolyline(3D)

NAME

DoLineWidth – Create a line width primitive attribute object

SYNOPSIS

C:

```
DtObject DoLineWidth(linewidth)
DtReal linewidth;
```

Fortran:

```
INTEGER*4 DOLW(LINWID)
REAL*8 LINWID
```

DESCRIPTION

DoLineWidth creates a line width primitive attribute object. The parameter *linewidth* is a scaling factor for the line width of lines created using Doré primitive object functions. It must be a positive, non-zero, floating point number.

DEFAULTS

The default *DoLineWidth* is 1.0.

SEE ALSO

DoLineList(3D), DoLineType(3D), DoPolyline(3D)

NAME

DoLookAtFrom – Create an orienting geometric tranformation object

SYNOPSIS

C:

```
DtObject DoLookAtFrom(at, from, up)
DtPoint3 at, from;
DtVector3 up;
```

Fortran:

```
INTEGER*4 DOLAF(AT, FROM, UP)
REAL*8 AT(3), FROM(3)
REAL*8 UP(3)
```

DESCRIPTION

DoLookAtFrom creates a geometric transformation object that modifies the current transformation matrix attribute. *DoLookAtFrom* specifies a modeling transformation that combines a translation and a series of rotations.

The parameter *from* specifies the position of the new origin. The ray between *at* and *from* specifies the new Z-direction. The parameter *up* specifies the new Y-direction as the projection of the *up* vector onto the new XY-plane. The new X-direction is computed such that the new X-, Y-, and Z-directions define a right-handed coordinate system.

DoLookAtFrom commonly specifies the position, direction of gaze, and the up direction of light and camera objects in a definition group. It may also be used to orient primitive objects within a scene.

ERRORS

The results of *DoLookAtFrom* are undefined if

The *at* and *from* points lie in a line parallel to the *up* vector.

The *at* and *from* points are identical.

The *up* vector is of 0 length.

[WARNING - degenerate lookatfrom specification]

SEE ALSO

DoRotate(3D), DoScale(3D), DoShear(3D), DoTransformMatrix(3D), DoTranslate(3D)

NAME

DoMarkerFont – Create a marker font primitive attribute object

SYNOPSIS

C:

```
DtObject DoMarkerFont(font)
DtFont font;
```

Fortran:

```
INTEGER*4 DOMF(FONT)
INTEGER*4 FONT
```

DESCRIPTION

DoMarkerFont creates a marker font primitive attribute object. The parameter *font* determines the font used for subsequent polymarker primitive objects.

DEFAULTS

The default *DoMarkerFont* is *DcSymbols <DCFSYM>*.

SEE ALSO

DoMarkerGlyph(3D), DoMarkerScale(3D), DoPolymarker(3D)

NAME

DoMarkerGlyph – Create a marker glyph primitive attribute object

SYNOPSIS

C:

```
DtObject DoMarkerGlyph(glyph)
DtInt glyph;
```

Fortran:

```
INTEGER*4 DOMG(GLYPH)
INTEGER*4 GLYPH
```

DESCRIPTION

DoMarkerGlyph creates a marker glyph primitive attribute object. The parameter *glyph* specifies the marker glyph for all following polymarker primitive objects. A marker glyph is an index into the marker font.

There is a special glyph that can be reached from any marker font using the index *DcMarkerPoint <DCMKPT>*. This glyph always represents itself as a single pixel on the screen, independent of marker scaling.

DEFAULTS

The default *DoMarkerGlyph* is *DcMarkerPoint <DCMKPT>*.

SEE ALSO

DoMarkerFont(3D), DoMarkerScale(3D), DoPolymarker(3D)

NAME

DoMarkerScale – Create a marker scale factor primitive attribute object

SYNOPSIS

C:

```
DtObject DoMarkerScale(scale)
DtReal scale;
```

Fortran:

```
INTEGER*4 DOMS(SCALE)
REAL*8 SCALE
```

DESCRIPTION

DoMarkerScale creates a marker scale factor primitive attribute object. The parameter *scale* specifies the relative size of the image of polymarker primitive objects. The actual size of the displayed marker is determined by multiplying the marker scale factor by a device-dependent nominal marker size; the result is in frame units. Polymarkers are represented by single points until rendering, and thus have no dimension (or scale) until the marker glyph is used. The parameter *scale* may be any floating point number; negative numbers will invert the maker representation in X and Y. If the parameter *scale* is 0, the marker will be a single point. The same result can be achieved with less overhead using the marker glyph *DcMarkerPoint* <*DCMKPT*>.

DEFAULTS

The default *DoMarkerScale* is 1.0.

SEE ALSO

DoMarkerFont(3D), DoMarkerGlyph(3D), DoPolymarker(3D)

NAME

DoMatrix – Create a matrix object

SYNOPSIS

C:

```
DtObject DoMatrix(n,m,data)
DtInt n, m;
DtReal data[];
```

Fortran:

```
INTEGER*4 DOM(N, M, DATA)
INTEGER*4 N, M
REAL*8 DATA(M, N)
```

DESCRIPTION

DoMatrix creates a matrix object. The object represents an *n* by *m* matrix, with values given in the *data* array argument. Matrix objects are useful as arguments to the *DoPatch* <*DOPAT*> routine. Doré provides a number of standard matrices, including:

DcBezier3 <*DCMXB3*> - quadratic Bezier curve basis.
DcBezier4 <*DCMXB4*> - cubic Bezier patch basis.
DcHermite4 <*DCMXH4*> - cubic Hermite patch basis.
DcBSpline4 <*DCMXBS4*> - cubic BSpline patch basis.

ERRORS

DoMatrix will fail if *n* or *m* is greater than 20.

[WARNING - value out of range]

SEE ALSO

DoPatch(3D)

NAME

DoMinBoundingVolExt – Create an attribute object which specifies the minimum renderable bounding extent

SYNOPSIS

C:

```
DtObject DoMinBoundingVolExt(extension)
DtReal Extension;
```

Fortran:

```
INTEGER*4 DOMBVE(EXTENS)
REAL*8 EXTENS
```

DESCRIPTION

DoMinBoundingVolExt creates an attribute object which specifies the minimum renderable bounding extent. The parameter *extension* specifies the shortest length in device coordinates (pixels) that a bounding volume (specified with *DoBoundingVol <DOBV>*) must span in X and Y on the device before the renderer will ignore objects. The renderer will either ignore objects below that bounding volume or render their alternate object instead.

The bounding volume object's intersection with the display volume is computed when a bounding volume object is executed. If the intersection is empty, that is if the bounding volume is not currently visible, then execution of the current group is aborted because nothing else in the group will draw. If the intersection is not empty, then a further test is made. First, the smallest right rectangle that can enclose the projection of the bounding volume on the device is computed. Then, if the length of the diagonal of that rectangle is less than the current minimum bounding extention, then execution of the current group will be aborted and any alternate object will be executed in its place.

See *DoBoundingVol* for more information on the process used to determine whether an object or its alternate representation should be rendered.

DEFAULTS

The default value for *DoMinBoundingVolExt* is 2.0 pixels.

SEE ALSO

DoBoundingVol(3D), DoBoundingVolSwitch(3D), DsCompBoundingVol(3D)

NAME

DoNURBSurf – Create a non-uniform rational B-Spline surface primitive object

SYNOPSIS

C:

```
DtObject DoNURBSurf(colormodel, ctrlpointtype, uv_area, order_u, n_knot_u,
        knot_u, order_v, n_knot_v, knot_v, n_ctrl_u, n_ctrl_v, ctrl_vertices)
DtColorModel colormodel;
DtCtrlPointType ctrlpointtype;
DtNArea uv_area;
DtInt order_u, order_v;
DtInt n_knot_u, n_knot_v;
DtReal knot_u[ ], knot_v[ ];
DtInt n_ctrl_u, n_ctrl_v;
DtReal ctrl_vertices[ ];
```

Fortran:

```
INTEGER*4 DONRBS(COLMOD, CPTTYP, UVAREA, ORDERU, NKNOTU,
        KNOTU, ORDERV, NKNOTV, KNOTV, NCTRLU, NCTRLV, CTLVTX)
INTEGER*4 COLMOD
INTEGER*4 CPTTYP
REAL*8 UVAREA(4)
INTEGER*4 ORDERU, ORDERV
INTEGER*4 NKNOTU, NKNOTV
REAL*8 KNOTU(NKNOTU), KNOTV(NKNOTV)
INTEGER*4 NCTRLU, NCTRLV
REAL*8 CTLVTX(4, NCTRLV, NCTRLU)
```

DESCRIPTION

DoNURBSurf defines a non-uniform rational BSpline surface primitive object.

The parameter *colormodel* specifies the color model used if the control points contain color information for shading purposes. The parameter *ctrlpointtype* specifies the nature of the control points.

The parameter *uv_area* consists of two pairs of *DtReal*s that define the parametric intervals used by the NURB surface. The parameters *n_knot_u* and *n_knot_v* specify the number of values in the lists *knot_u* and *knot_v*. The parameters *order_u* and *order_v* express the overall order of the NURB along u and v. The parameters *n_ctrl_u* and *n_ctrl_v* specify the layout of vertices as a grid. The vertices themselves (*ctrl_vertices*) are contained in a one-dimensional array of four dimensional control points. They are ordered in the array the same as if they were being read as sequencial values from a two-dimensional array of u and v with v varying fastest. Note that the C array element foo[i][j] is equivalent to the Fortran array element FOO(j+1,i+1).

If this primitive object defines a closed surface or will be oriented with backfacing parts of the surface away from the viewer, then backface culling may speed up the rendering time. See *DoBackfaceCullable* and *DoBackfaceCullSwitch*.

ERRORS

n_knot_u must equal the sum of *order_u* and *n_ctrl_u*, and *n_knot_v* must equal the sum of *order_v* and *n_ctrl_v*.

[WARNING - value out of range]

DoNURBSurf will fail if *order_u* and *order_v* and not between 0 and 25, inclusive.

[SEVERE - bad curve order]

SEE ALSO

DoBackfaceCullSwitch(3D), DoBackfaceCullable(3D), DoPatch(3D), DoRepType(3D), DoSubDivSpec(3D), DoInterpType(3D)

NAME

DoNameSet – Create a nameset modifying object

SYNOPSIS

C:

```
DtObject DoNameSet(n, members, setop)
DtInt    n;
DtInt    members[];
DtSetOperation setop;
```

Fortran:

```
INTEGER*4 DONS(N, MEMBER, SETOP)
INTEGER*4 N
INTEGER*4 MEMBER(N)
INTEGER*4 SETOP
```

DESCRIPTION

DoNameSet creates an organizational object for modifying nameset attributes. Nameset attributes, in conjunction with filter attributes, provide a high-level control over the invisibility and pickability. See *DoFilter* for details.

The parameter *members* is a list of members of the specified nameset to be affected. *n* is the number of elements in the list. The argument *setop* specifies how the status of the members referred to in *members* is to be affected. The valid choices for *setop* are:

DcSetAdd <DCSADD>
Adds the members listed in *members* to the current nameset.

DcSetDelete <DCSDEL>
Removes the members listed in *members* from the current nameset.

DcSetInvert <DCSINV>
Causes all members of the nameset listed in *members* to take on the opposite of their current values. In other words, if the given set members are not in the current nameset, they will be added; if they are included, they will be removed.

DcSetReplace <DCSREP>
Replaces the current value of the specified nameset with one that includes only the members listed in *members*.

ERRORS

DoNameSet will fail if an invalid *setop* value is specified.

[FATAL - invalid set operation]

DEFAULTS

The default nameset is empty.

SEE ALSO

DoFilter(3D), DoInvisSwitch(3D), DoPickSwitch(3D)

NAME

DoParallel – Create a parallel studio attribute object for cameras

SYNOPSIS

C:

```
DtObject DoParallel(window_size, hither, yon)
DtReal window_size, hither, yon;
```

Fortran:

```
INTEGER*4 DOPAR(WWSIZE, HITHER, YON)
REAL*8 WWSIZE, HITHER, YON
```

DESCRIPTION

DoParallel creates a parallel studio attribute object for cameras. The parameter *window_size* specifies the square size of the field of view that is centered at the origin and parallel to the XY-plane. The parameters *hither* and *yon* specify the location of the front and the back clipping planes in camera coordinates. Note that, for parallel cameras, these planes may actually be behind the camera (that is, positive numbers).

ERRORS

DoParallel will fail if *hither* is less than or equal to *yon*

[WARNING - value out of range]

SEE ALSO

DoCameraMatrix(3D), DoPerspective(3D), DoProjection(3D)

NAME

DoPatch – Create a patch primitive object

SYNOPSIS

C:

```
DtObject DoPatch(colormodel, vertextype, matrixl, vertices, matrixr)
DtColorModel colormodel;
DtVertexType vertices;
DtObject matrixl;
DtObject matrixr;
DtReal vertices[];
```

Fortran:

```
INTEGER*4 DOPAT(COLMOD, VTXTYP, MATRXL, VTXS, MATRXR)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 MATRXL
INTEGER*4 MATRXR
REAL*8 VTXS(3,*)
```

DESCRIPTION

DoPatch creates a patch primitive object. The patch type is determined by the pair of matrices used. These matrices could be *DcBezier3* <*DCMXB3*>, *DcBezier4* <*DCMXB4*>, *DcHermite4* <*DCMXH4*>, or *DcBSpline4* <*DCBSP4*>. The user may specify a non-standard matrix object with *DoMatrix* <*DOM*>.

The parameter *colormodel* specifies the color model if the control points specify color for shading purposes. The parameter *vertextype* specifies the nature of the control points; only types *DcLoc* <*DCL*> and *DcLocClr* <*DCLC*> are allowed. See the appendix on vertex types for more information.

The patch is defined by the list of control points in the parameter *vertices*. The area for the patch control points in control space is always 0.0 to 1.0 in each direction.

By allowing different matrices to be specifed for both the left (*matrixl*) and right (*matrixr*) matrices, both the size and type of parameterization in each direction can be individually controlled. If the same matrix type is specified for both matrices, then the entire patch will be of that type; the right matrix is internally transposed before use. The order is determined by the dimension of each matrix, i.e, the number of vertices in that direction. The degree will always be one less than this number. Note that the C array element foo[i][j] is equivalent to the Fortran array element FOO(j+1,i+1).

The use of the matrices follows this equation:

$$P(u,v) = [1\ u\ u^2\ u^3\ \ldots] * matrixl * vertices * matrixr^T * [1\ v\ v^2\ v^3\ \ldots]^T$$

If this primitive object defines a closed surface or will be oriented with backfacing parts of the surface away from the viewer, then backface culling may speed up the rendering time. See *DoBackfaceCullable* and *DoBackfaceCullSwitch*.

ERRORS

DoPatch will fail if the vertex type is not *DcLoc* or *DcLocClr*.

[WARNING - bad vertex type]

DoPatch will fail if an invalid matrix object is specified.

[WARNING - invalid matrix object handle]

DoPatch will fail if a non-square matrix is specified.

[WARNING - non square matrix]

SEE ALSO

DoBackfaceCullSwitch(3D), DoBackfaceCullable(3D), DoNURBSurf(3D), VertexTypes(3D)

NAME

DoPerspective – Create a perspective studio attribute object for cameras

SYNOPSIS

C:

```
DtObject DoPerspective(fov, hither, yon)
DtReal fov, hither, yon;
```

Fortran:

```
INTEGER*4 DOPER(FOV, HITHER, YON)
REAL*8 FOV, HITHER, YON
```

DESCRIPTION

DoPerspective creates a perspective studio attribute object for cameras. The parameter *fov* specifies the frustum of vision angle (field of view) in degrees. The parameters *hither* and *yon* specify the location of the front and the back clipping planes in camera coordinates. Note that both of these planes must be ahead of the camera. Because the camera's location in its own relative coordinate system is always at (0.0, 0.0, 0.0) looking down the negative Z-axis, both *hither* and *yon* must be negative.

ERRORS

DoPerspective will fail if *hither* is less than or equal to *yon.*

[WARNING - value out of range]

DoPerspective will fail if *hither* is greater than or equal to 0.0.

[WARNING - value out of range]

SEE ALSO

DoCameraMatrix(3D), DoParallel(3D), DoProjection(3D)

NAME

DoPickID – Create a pick identifier object

SYNOPSIS

C:

```
DtObject DoPickID(pickid)
DtInt pickid;
```

Fortran:

```
INTEGER*4 DOPID(PICKID)
INTEGER*4 PICKID
```

DESCRIPTION

DoPickID returns a pick identifier organizational object. The argument *pickid* determines the pick identification for subsequent objects. A pick identification is an arbitrary 32-bit value that is used to identify various parts of the Doré database. It is similar to a nameset in its ability to identify or give special meaning to parts of the database.

The pick identifications are part of the information returned in a pick. They are returned for the picked primitives as well as for their ancestors in the hierarchy.

DEFAULTS

The default *DoPickID* is 0.

SEE ALSO

DdPickObjs(3D), DoNameSet(3D), DdSetPickCallback(3D)

NAME

DoPickSwitch – Create a pickability switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoPickSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOPS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoPickSwitch creates a pickability primitive attribute object. Executing this object sets the current value of the pickability switch to *switchvalue*. Possible values for *switchvalue* are:

DcOn <DCON>
Subsequent objects are elegible to be picked.

DcOff <DCOFF>
Subsequent objects will be ignored for picking.

DoPickSwitch provides direct control of the pickability primitive attribute at the lowest level. For higher level control, see *DoNameSet* and *DoFilter*.

DEFAULTS

The default *DoPickSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DdPickObjs(3D), DoNameSet(3D), DoFilter(3D)

NAME

DoPointList – Create a point list primitive object

SYNOPSIS

C:

```
DtObject DoPointList(colormodel, vertextype, pointcount, vertices)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt pointcount;
DtReal vertices[];
```

Fortran:

```
INTEGER*4 DOPNTL(COLMOD, VTXTYP, PNTCNT, VTXS)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 PNTCNT
REAL*8 VTXS(*,*)
```

DESCRIPTION

DoPointList creates a primitive object that defines a list of independent points. The parameter *colormodel* specifies the color model to be used when defining vertex colors (if any). The parameter *vertextype* specifies the exact nature of the vertex: whether the vertex contains color and/or normal information for shading purposes. See the appendix on vertex types for more information on this parameter. The parameter *pointcount* contains the number of points in the list. The parameter *vertices* is an array containing data for *pointcount* vertices.

If vertex normals are not provided, backface culling has no effect for points because they have no surface on which to define a geometric normal.

SEE ALSO

DoLineList(3D), VertexTypes(3D)

NAME

DoPolygon – Create a polygon primitive object

SYNOPSIS

C:

```
DtObject DoPolygon(colormodel, vertextype, contourcount, contours,
  vertices, shape)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt contourcount;
DtInt contours[contourcount];
DtReal vertices[];
DtShapeType shape;
```

Fortran:

```
INTEGER*4 DOPGN(COLMOD, VTXTYP, CNTCNT, CONTURS, VTXS, SHAPE)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 CNTCNT
INTEGER*4 CONTURS(CNTCNT)
REAL*8 VTXS(*)
INTEGER*4 SHAPE
```

DESCRIPTION

DoPolygon creates a general polygon primitive object. A general polygon consists of a number of contours comprising the outer boundary (or boundaries) plus any interior holes and/or islands. Contours may be self-intersecting and may intersect each other. The polygon is assumed to be planar.

The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *vertices* is a one-dimensional array of vertex data.

The parameter *contourcount* specifies the number of contours in the polygon. The parameter *contours* is an array of *contourcount* integers, each of which specifies the number of vertices in the associated contour. The first contour in the polygon begins with vertex 0 in the *vertices* array and continues for the length (in vertices) of the contour. The last vertex of a contour is implicitly connected to its first vertex. The next contour in the polygon begins with the next vertex in the *vertices* array.

The parameter *shape* hints at the geometry of the polygon. This argument has meaning only for polygons that consist of a single contour, i.e., whose *contourcount* is 1. For these polygons, correctly specifying the shape of the contour will result in optimal polygon decomposition when necessary. Incorrectly specifying the shape may result in an incorrect or aborted decomposition. The possible values for *shape* are:

DcConvex <DCCNVX>
Path is wholly convex.

DcConcave <DCCNCV>
Path may be concave but not self-intersecting.

DcComplex <DCCPLX>
Path may be self-intersecting.

The geometric normal to the polygon is calculated using the right-hand rule (the fingers of the right hand point in the order of vertices and the thumb points in the direction of the normal). This normal is used to determine the way that the polygon

faces for backface culling and the shading when the polygon is faceted (no vertex normals provided) or facet shading has been specified.

ERRORS

DoPolygon must be called with a valid specification for *shape.*

[WARNING - invalid parameter]

DoPolygon must be called with at least one contour of three vertices.

[WARNING - insufficient information]

SEE ALSO

DoPolygonMesh(3D), DoSimplePolygon(3D), DoSimplePolygonMesh(3D), VertexTypes(3D)

NAME

DoPolygonMesh – Create a polygon mesh primitive object

SYNOPSIS

C:

```
DtObject DoPolygonMesh(colormodel, vertextype, vertexcount, vertices,
  polygoncount, polygons, contours, vertexlist, shape, smoothflag)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt vertexcount;
DtReal vertices[];
DtInt polygoncount;
DtInt polygons[];
DtInt contours[];
DtInt vertexlist[];
DtShapeType shape;
DtFlag smoothflag;
```

Fortran:

```
INTEGER*4 DOPGNM(COLMOD, VTXTYP, VTXCNT, VTXS,
                 PLYCNT, PLYGON, CONTURS, VTXLST,
                 SHAPE, SMOOFL)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 VTXCNT
REAL*8 VTXS(*)
INTEGER*4 PLYCNT
INTEGER*4 PLYGON(PLYCNT)
INTEGER*4 CONTURS(*)
INTEGER*4 VTXLST(*)
INTEGER*4 SHAPE
INTEGER*4 SMOOFL
```

DESCRIPTION

DoPolygonMesh creates a primitive object that defines a collection of general polygons, normally interconnected. Each polygon must be as described in *DoPolygon*.

Mesh objects allow the user to specify a collection of vertices in space and connect the vertices into a collection of other geometric entities, usually forming a connected surface. Polygon meshes are generally used to approximate a smooth surface.

Geometric normals to the polygons in the mesh are calculated using the right-hand rule (the fingers of the right hand point in the order of vertices and the thumb points in the direction of the normal). These normals are used to determine the way polygons face for backface culling and the shading when the polygons are faceted (no vertex normals provided and smoothflag off) or facet shading has been specified. If this primitive object defines a closed surface or is oriented such that parts of it face away from the viewer, backface culling may speed the rendering time.

The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *vertices* is an array of vertex data. The parameter *vertexcount* specifies the number of vertices contained in the *vertices* array.

The parameter *polygoncount* specifies the number of polygons in the mesh. The parameter *polygons* is an array of *polygoncount* integers, each of which specifies the number of contours in the associated polygon. The parameter *contours* is an array of

integers, one for each of the total number of contours in the mesh. Each entry in the *contours* array specifies the number of verticies in the associated contour.

The parameter *vertexlist* is an array of integers that index the *vertices* array. The first polygon in the mesh begins with the vertex specified by *vertexlist[0]*, and continues using the vertices specified by the following entries in *vertexlist*. The last vertex of a contour is implicitly connected to its first vertex. The next polygon begins with the vertex specified next in the *vertexlist* array.

The parameter *shape* hints at the geometry of the polygons in the mesh. This argument has meaning only for polygons that consist of a single contour, i.e., whose *contourcount* is 1. For these polygons, correctly specifying the shape of the contour will result in optimal polygon decomposition when necessary. Incorrectly specifying the shape may result in an incorrect or aborted decomposition. The possible values for *shape* are:

DcConvex <*DCCNVX*>
Path is wholly convex.

DcConcave <*DCCNCV*>
Path may be concave but not self-intersecting.

DcComplex <*DCCPLX*>
Path may be self-intersecting.

The parameter *smoothflag* specifies that, if vertex normals are not provided, the average of the geometric normals from a vertex's adjacent surfaces constitute the normal for shading. (This assumes the polygon mesh defines a smooth surface.)

ERRORS

DoPolygonMesh must be called with a valid specification for *shape*.

[WARNING - invalid parameter]

DoPolygonMesh must be called with at least one contour of three vertices.

[WARNING - insufficient information]

DoPolygonMesh will issue a warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

SEE ALSO

DoPolygon(3D), DoSimplePolygon(3D), DoSimplePolygonMesh(3D), VertexTypes(3D)

NAME

DoPolyline – Create a connected line segment primitive object

SYNOPSIS

C:

```
DtObject DoPolyline(colormodel, vertextype, vertexcount, vertices)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt vertexcount;
DtReal vertices[];
```

Fortran:

```
INTEGER*4 DOPL(COLMOD, VTXTYP, VTXCNT, VTXS)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 VTXCNT
REAL*8 VTXS(*)
```

DESCRIPTION

DoPolyline creates a primitive object that defines a collection of vertices connected into a continuous set of line segments.

The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *vertexcount* contains the number of vertices; there must be at least two. The parameter *vertices* is an array of vertices. Vertices are listed in the order in which they connect. Polylines may be circular, interconnected, or may reconnect the same pair of vertices multiple times.

ERRORS

DoPolyline will error if *vertexcount* is less than two.

[WARNING - invalid parameter]

SEE ALSO

DoLineList(3D), DoLineWidth(3D), DoLineType(3D), VertexTypes(3D)

NAME

DoPolymarker – Create a polymarker primitive object

SYNOPSIS

C:

```
DtObject DoPolymarker(markercount, markerpositions)
DtInt markercount;
DtPoint3 markerpositions[];
```

Fortran:

```
INTEGER*4 DOPM(MKRCNT, MKRPOS)
INTEGER*4 MKRCNT
REAL*8 MKRPOS(3,MKRCNT)
```

DESCRIPTION

DoPolymarker creates a polymarker primitive object. A polymarker primitive object is a set of markers, each having its own location.

The locations of the markers are specified by an array of points. The parameter *markercount* specifies the number of points in the array *markerpositions*. The parameter *markerpositions* is the two-dimensional array of marker data. The array will contain 3 real numbers for each marker.

A marker has no geometric size, so transformations will not affect the displayed size of a marker. A marker is drawn using a glyph that can be specified using *DoMarkerGlyph <DOMG>* from a font that can be specified by *DoMarkerFont <DOMF>*. The glyph can be scaled by a factor that can be set by *DoMarkerScale <DOMS>*.

SEE ALSO

DoMarkerFont(3D), DoMarkerGlyph(3D), DoMarkerScale(3D)

NAME

DoPopAtts – Create an organizational object that pops all attributes

SYNOPSIS

C:

DtObject DoPopAtts()

Fortran:

INTEGER*4 DOPPA()

DESCRIPTION

DoPopAtts creates an organizational object that pops all attributes. This object restores the local attribute environment that was saved with *DoPushAtts* <*DOPUA*>. See *DoPushAtts* for details on using these functions.

ERRORS

DoPopAtts will generate a warning if it is invoked in a group without a corresponding *DoPushAtts* <*DOPUA*>.

This warning is only generated during traversal of the database, not during creation of the object. For example, the warning will not occur when the initial code that creates the object is executed; rather when a display update or other traversal occurs.

[WARNING - cannot pop stack]

SEE ALSO

DoGroup(3D), DoInLineGroup(3D), DoPopMatrix(3D), DoPushAtts(3D), DoPushMatrix(3D)

NAME

DoPopMatrix – Create an organizational object that pops a matrix

SYNOPSIS

C:

DtObject DoPopMatrix()

Fortran:

INTEGER*4 DOPPMX()

DESCRIPTION

DoPopMatrix creates an organizational object that pops a matrix. This object restores a local environment of geometric transformations saved with *DoPushMatrix* <*DOPUMX*>. See *DoPushMatrix* for details on using these functions.

ERRORS

DoPopMatrix will generate a warning if it is invoked in a group without a corresponding *DoPushMatrix* <*DOPUMX*>.

This warning is generated during traversal of the database, not during creation of the object. The warning will not occur when the initial code that creates the object is executed; rather the warning will occur when a display update or other traversal occurs.

[WARNING - cannot pop matrix]

SEE ALSO

DoGroup(3D), DoInLineGroup(3D), DoPopAtts(3D), DoPushAtts(3D), DoPushMatrix(3D)

NAME

DoPrimSurf – Create a primitive surface primitive object

SYNOPSIS

C:

```
DtObject DoPrimSurf(surfacetype)
DtSurface surfacetype;
```

Fortran:

```
INTEGER*4 DOPMS(SRFTYP)
INTEGER*4 SRFTYP
```

DESCRIPTION

DoPrimSurf creates a primitive surface primitive object. A primitive surface is the outer skin of a classic volume like a sphere, box, or cylinder.

Primitive surfaces are always defined in a standard local orientation. The user can change their form by applying geometric transformations.

The parameter *surfacetype* specifies the type of primitive surface. Possible values are:

DcSphere <DCSPHR>
: Sphere centered at the origin of the object's local coordinate system with a radius of 1.0.

DcCylinder <DCCYL>
: Cylinder with front flat circular plate in the XY-plane of the object's local coordinate system and a radius of 1.0. The cylinder is 1.0 unit deep and extends in the positive Z-direction.

DcBox <DCBOX>
: Box with front face in the XY-plane of the object's local coordinate system. Its bottom left corner is at the origin and it extends <1.0> unit in the positive X-, Y-, and Z-directions.

DcCone <DCCONE>
: Cone with the base circular plate in the XY-plane of the object's local coordinate system and a radius of 1.0. The apex of the cone extends 1.0 unit in the positive Z-direction.

For some rendering states, a primitive surface is decomposed into triangles before it is rendered. The amount of decomposition performed is determined by the *DoSubDivSpec <DOSDS>* attribute.

When an object defines a closed surface or is oriented such that parts of it face away from the viewer, backface culling will speed up the rendering time.

ERRORS

DoPrimSurf will fail if an invalid primitive surface type is specified.

[WARNING - value out of range]

SEE ALSO

DoBackfaceCullSwitch(3D), DoBackfaceCullable(3D), DoSubDivSpec(3D)

NAME

DoProjection – Create a projection studio attribute object for cameras

SYNOPSIS

C:

```
DtObject DoProjection(window, ptype, prp, viewplane, hither, yon)
DtArea *window;
DtProjectionType ptype;
DtPoint3 prp;
DtReal viewplane;
DtReal hither;
DtReal yon;
```

Fortran:

```
INTEGER*4 DOPRJ(WINDOW, PTYPE, PRP, VWPLAN, HITHER, YON)
REAL*8 WINDOW(2)
INTEGER*4 PTYPE
REAL*8 PRP(3)
REAL*8 VWPLAN
REAL*8 HITHER
REAL*8 YON
```

DESCRIPTION

DoProjection creates a projection studio attribute object for cameras. The parameters *hither* and *yon* specify the location of the near and far clipping planes relative to the camera. The hither plane, yon plane, and viewing plane are defined perpendicular to the direction of view and should be specified in camera coordinates. The parameter *viewplane* specifies the distance from the camera to the viewing plane. The parameter *window* specifies an arbitrary rectangular region on the viewing plane.

The parameter *prp* specifies a point relative to the camera. If the parameter *ptype* is *DcPerspective* <*DCPRSP*>, *prp* and the window corners specify a pyramid. The region of the pyramid between the near and far clipping planes defines the view volume. If *ptype* is *DcParallel* <*DCPRLL*>, the view volume is the parallelpiped produced by the parallel projection of the window corners between the near and far clipping planes along the line from the *prp* through the center of the window. In both cases, the resulting view volume is mapped into the rectangular volume described by the view volume.

Because the camera's location (in its own relative coordinate system) is always (0.0, 0.0, 0.0) looking down the negative Z-axis, both the hither plane and the yon plane must be ahead of the camera, i.e., the parameters *hither* and *yon* must be negative.

ERRORS

DoProjection will fail if *hither* is less than or equal to *yon*.

[WARNING - value out of range]

DoProjection will fail if *hither* is greater than or equal to 0.0.

[WARNING - value out of range]

SEE ALSO

DoCameraMatrix(3D), DoParallel(3D), DoPerspective(3D)

NAME

DoPushAtts – Create an organizational object that pushes all attributes

SYNOPSIS

C:

DtObject DoPushAtts()

Fortran:

INTEGER*4 DOPUA()

DESCRIPTION

DoPushAtts creates an organizational object that pushes all attributes. This object saves the local environment of all attributes; the environment is restored with *DoPopAtts <DOPPA>*.

DoPushAtts and *DoPopAtts <DOPPA>* act as a team to create a local environment of attributes that may be saved and restored. *DoPushAtts* effectively pushes onto a local stack any attributes that could be modified by objects between its location in the regular group and a subsequent *DoPopAtts* object. Execution of the *DoPopAtts <DOPPA>* object restores the values from the stack of all attributes values that were pushed. *DoPushAtts* and *DoPopAtts <DOPPA>* objects may be nested; however, *DoPushAtts* and *DoPopAtts <DOPPA>* will not work across group boundaries.

During execution, regular groups act as if they performed a *DoPushAtts* before their execution starts and a *DoPopAtts <DOPPA>* when they are exited. Inline groups behave differently; see *DoInLineGroup*. When a group is closed, all *DoPushAtts* still in effect are popped.

SEE ALSO

DoGroup(3D), DoInLineGroup(3D), DoPopAtts(3D), DoPopMatrix(3D), DoPushMatrix(3D)

NAME

DoPushMatrix – Create an organizational object that pushes a matrix

SYNOPSIS

C:

DtObject DoPushMatrix()

Fortran:

INTEGER*4 DOPUMX()

DESCRIPTION

DoPushMatrix creates an organizational object that pushes a matrix. This object saves a local environment of geometric transformations; the environment is restored by *DoPopMatrix <DOPPMX>*.

DoPushMatrix and *DoPopMatrix <DOPPMX>* act as a team to create a local environment of geometric transformations that can be saved and restored. *DoPushMatrix* pushes onto a local stack the transformation matrix that could be modified by objects between its location in the regular group and a subsequent *DoPopMatrix <DOPPMX>* object. Execution of the *DoPopMatrix* object pops the transformation matrix from the stack thus restoring the transformation matrix to its earlier value. Any geometric transformations encountered between the *DoPushMatrix* and *DoPopMatrix <DOPPMX>* objects are lost. *DoPushMatrix* and *DoPopMatrix <DOPPMX>* group objects may be nested; however, *DoPushMatrix* and *DoPopMatrix <DOPPMX>* will not work across group boundaries.

During execution, regular groups act as if they performed a *DoPushMatrix* before their execution starts and a *DoPopMatrix <DOPPMX>* when they are exited. Inline groups behave differently; see *DoInLineGroup*. When a group is closed, all *DoPushMatrix* objects still in effect will be popped.

To perform a series of geometric transformations relative to an absolute position, use *DoPushMatrix* before each set of geometric transformations and *DoPopMatrix* afterward. This makes each set of geometric transformations relative to the original starting position.

The transformation matrix attribute is also saved and restored with *DoPushAtts <DOPUA>* and *DoPopAtts <DOPPA>*.

SEE ALSO

DoGroup(3D), DoInLineGroup(3D), DoPopAtts(3D), DoPushAtts(3D), DoPushMatrix(3D)

NAME

DoRaster – Create a raster object

SYNOPSIS

C:

```
DtObject DoRaster(width, height, depth, type, typestring, data,
                  delcallback)
DtInt width;
DtInt height;
DtInt depth;
DtRasterType type;
DtPtr typestring;
DtPtr data;
DtObject delcallback;
```

Fortran:

```
INTEGER*4
  DORS(WIDTH, HEIGHT, DEPTH, TYPE, TYPSTR, TLEN, DATA,
       DELCBK)
INTEGER*4 WIDTH
INTEGER*4 HEIGHT
INTEGER*4 DEPTH
INTEGER*4 TYPE
INTEGER*4 TLEN
CHARACTER*TLEN TYPSTR
INTEGER*4 DATA
INTEGER*4 DELCBK
```

DESCRIPTION

DoRaster creates a raster object. Raster objects can, for example, be used as texture maps or as backgrounds for views.

The parameter *data* is a pointer to the actual raster data in memory. The data may be for a one-, two- or three-dimensional raster. The parameters *width, height,* and *depth* specify the dimensions of the raster. Note that Doré does not copy the raster data pointed to by *data,* but uses it from the user space. The data in user space can be modified at any time, but Doré must be notified (See *DsRasterUpdate*).

The parameter *type* indicates the type and format of the raster data. Possible values for *type* are:

DcRasterRGB <DCRRGB>
Each point in the raster has red, green and blue information.

DcRasterRGBA <DCRRA>
Each point in the raster has red, green, blue and alpha information.

DcRasterRGBAZ <DCRRAZ>
Each point in the raster has red, green, blue, alpha and z information.

DcRasterRGBZ <DCRRZ>
Each point in the raster has red, green, blue and z information.

DcRasterA <DCRA>
Each point in the raster has alpha information.

DcRasterZ <DCRZ>
Each point in the raster has z information.

DcRasterSpecial <DCRSPC>
The data type is not one of the standard Doré types.

The standard Doré types assume that the data is pixel interleaved. If a renderer requires a format that is not one of the standard Doré ones, then the type should be set to be *DcRasterSpecial <DCRSPC>*. In that case, the character string *typestring* specifies the format of the data. The parameter *typestring* is ignored if the *type* is not *DcRasterSpecial <DCRSPC>*.

The parameter *delcallback* is a callback object. When the raster object created by *DoRaster* is deleted by Doré the function defined by *delcallback* will be invoked. A callback function takes the following form:

```
my_delcallback_fcn(data, ras_dataptr)
DtPtr data;      /* if data is a pointer to data */
Dt32bits data;   /* if data is a value */
DtPtr ras_dataptr; /* pointer to raster data */
```

Often an application will not need the data pointed to by *data* after the raster object is destroyed. Doré provides a standard callback object, *DcDeleteData <DCDELD>*, which will deallocate the space pointed to by *data*. The application can define its own callback if it does not want the data to be unconditionally deallocated when this raster object is deleted, for example, if the same data is used by other raster objects

FORTRAN SPECIFIC

The parameter *TYPSTR* is an optional format string that is *TLEN* characters long.

ERRORS

DoRaster will fail if passed an invalid delete callback object.

[WARNING - invalid callback object]

SEE ALSO

DoFileRaster(3D), DsRasterUpdate(3D), DoRasterWrite(3D)

NAME

DoReflectionSwitch – Create a reflection switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoReflectionSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOREFS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoReflectionSwitch creates a reflection switch primitive attribute object.

Reflection is the response of a surface to light from primitive objects in the environment where the reflected light is centered around the mirror direction. In computing the reflection component of a surface shade, a surface's response to reflected light will be determined by the specular color and the specular intensity attributes. Hence, the reflection color of the object will appear the same as its specular color except that its response is only to light reflected from primitive objects and not from light sources. Note also that the specular factor (*DoSpecularFactor* <*DOSPCF*>) is not used for reflection component calculations.

The parameter *switchvalue* specifies whether the surfaces of subsequently executed primitive objects have a reflection component contribution. Possible values are:

DcOff <*DCOFF*>
: The object will render itself without a reflection shading component regardless of the settings of the specular color and specular intensity attributes. These attributes will continue to be updated even though they will be ignored.

DcOn <*DCON*>
: The object will render itself using reflections whenever possible.

DEFAULTS

The default value for *DoReflectionSwitch* is *DcOff* <*DCOFF*>.

SEE ALSO

DoGlbRendRayLevel(3D), DoSpecularColor(3D), DoSpecularIntens(3D)

NAME

DoRefractionIndex – Create a refraction index primitive attribute object

SYNOPSIS

C:

```
DtObject DoRefractionIndex(index)
DtReal index;
```

Fortran:

```
INTEGER*4 DORFRI(INDEX)
REAL*8 INDEX
```

DESCRIPTION

DoRefractionIndex creates a refraction index primitive attribute object

When light passes from one material to another the path of the light ray is altered by refraction. This can be seen, for example, when light passes from air to a transparent material, such as glass or water.

How much a light ray is bent when it passes from one material to another depends on the ratio of the indices of refraction for the two materials. The parameter *index* is an index of refraction to be used for all subsequent primitive objects.

A primitive object will refract light rays only if the *DoRefractionSwitch* <*DORFRS*> is enabled.

DEFAULTS

The default *DoRefractionIndex* is 1.0

SEE ALSO

DoRefractionSwitch(3D)

NAME

DoRefractionSwitch – Create a refraction switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoRefractionSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DORFRS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoRefractionSwitch creates a refraction switch primitive attribute object

When light passes from one material to another the path of the light ray is altered by refraction. This can be seen, for example, when light passes from air to a transparent material, such as glass or water.

The parameter *switchvalue* specifies whether subsequent primitive objects can alter the path of light rays that pass through them.

DcOn <DCON>
The object alters the path of light rays passing through it.

DcOff <DCOFF>
The object never alters the path of light rays passing through it.

The *DoRefractionSwitch* attribute is ignored if the *DoTranspSwitch <DOTS>* is not turned on.

DEFAULTS

The default *DoRefractionSwitch* is *DcOff*.

SEE ALSO

DoTranspSwitch(3D), DoRefractionIndex(3D)

NAME

DoRepType – Create a representation type primitive attribute object

SYNOPSIS

C:

```
DtObject DoRepType(type)
DtRepType type;
```

Fortran:

```
INTEGER*4 DOREPT(TYPE)
INTEGER*4 TYPE
```

DESCRIPTION

DoRepType creates a representation type primitive attribute object. The *type* parameter specifies the primary geometric representation with which subsequent primitive objects will render themselves. Possible values are:

DcPoints <DCPNTS>
Points

DcWireframe <DCWIRE>
Outlines

DcSurface <DCSURF>
Surfaces

DcOutlines <DCOUTL>
Polygon outlines

DEFAULTS

The default value for *DoRepType* is *DcWireframe <DCWIRE>*.

SEE ALSO

DoInterpType(3D)

NAME

DoRotate – Create a rotation geometric transformation object

SYNOPSIS

C:

```
DtObject DoRotate(axis, radians)
DtAxis axis;
DtReal radians;
```

Fortran:

```
INTEGER*4 DOROT(AXIS, RADIAN)
INTEGER*4 AXIS
REAL*8 RADIAN
```

DESCRIPTION

DoRotate creates a rotation geometric transformation object that modifies the current transformation matrix attribute.

The parameter *axis* specifies the coordinate axis about which the rotation is to take place. Possible values are:

DcXAxis <*DCXAX*>

DcYAxis <*DCYAX*>

DcZAxis <*DCZAX*>

The parameter *radians* specifies the angle of rotation. It may take any positive or negative floating point value. Positive rotations are determined by the right-hand rule; when looking down a given axis from the direction of positive infinity towards the origin, positive rotations are counterclockwise.

SEE ALSO

DoLookAtFrom(3D), DoScale(3D), DoShear(3D), DoTransformMatrix(3D), DoTranslate(3D)

NAME

DoSampleAdaptive – Create an adaptive sampling camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleAdaptive(variance)
DtReal variance;
```

Fortran:

```
INTEGER*4 DOSADP(VARNCE)
INTEGER*4 VARNCE
```

DESCRIPTION

DoSampleAdaptive creates an adaptive sampling camera attribute object for antialiasing.

Adaptive sampling means that not all pixels will be sampled equally, with pixels that include object edges being sampled more frequently. Pixels will be sampled until the variance of the samples is below a certain treshold. Adaptive sampling can also be done in conjunction with supersampling.

The parameter *variance* is a number specifying the acceptable level of variance for a pixel. This number is typically between *0.0* and *1.0*. A higher number indicates a higher tolerance for aliases.

Adaptive sampling is enable/disabled with the *DoSampleJitterSwitch* <*DOSJSW*> attribute.

DEFAULTS

The default *DoSampleAdaptive* is 0.5.

SEE ALSO

DoSampleAdaptiveSwitch(3D), DoSampleSuper(3D)

NAME

DoSampleAdaptiveSwitch – Create an adaptive sampling switch camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleAdaptiveSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOSASW(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoSampleAdaptiveSwitch creates an adaptive sampling switch camera attribute object for antialiasing.

Adaptive sampling means that not all pixels will be sampled equally, with pixels that include object edges being sampled more frequently. Pixels will be sampled until the variance of the samples is below a certain treshold. Adaptive sampling can also be done in conjunction with supersampling.

The parameter *switchvalue* specifies whether subsequent cameras can sample adaptively.

DcOn <DCON>
Adaptive sampling is enabled.

DcOff <DCOFF>
Adaptive sampling is disabled.

DEFAULTS

The default *DoSampleAdaptiveSwitch* is *DcOff*.

SEE ALSO

DoSampleAdaptive(3D), DoSampleSuper(3D)

NAME

DoSampleFilter – Create a supersampling filter camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleFilter(filter, xwidth, ywidth)
DtObject filter;
DtReal xwidth;
DtReal ywidth;
```

Fortran:

```
INTEGER*4 DOSFLT(FILTER, XWIDTH, YWIDTH)
INTEGER*4 FILTER
REAL*8 XWIDTH
REAL*8 YWIDTH
```

DESCRIPTION

DoSampleFilter creates a supersampling filter camera attribute object for antialiasing.

Supersampling means that each pixel of an image is subdivided into subpixels, and a shade is computed for each of the subpixels. A filter is then applied to the shades of the subpixels to obtain the pixel value.

The computed pixel value is a weighted average of the subpixel shades, typically with higher weights for suppixels closer to the pixel center. The parameter *filter* specifies which filter function to use. Possible values for *filter* are:

DcFilterBox <DCFBOX>
: A box filter function is used. All subpixels under the filter footprint have equal weights.

The parameters *xwidth* and *ywidth* specify the size of the filter footprint in output pixels. For example, if *xwidth* and *ywidth* are both *1.0* then the filter footprint covers all the subpixels of a pixel. If *xwidth* and/or *ywidth* are larger than *1.0* then the filters will overlap.

DEFAULTS

The default *DoSampleSuperFilter* is *(DcFilterBox, 1., 1.)*.

SEE ALSO

DoSampleSuper(3D), DoSampleSuperSwitch(3D)

NAME

DoSampleJitter – Create a jitter sampling camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleJitter(factor)
DtReal factor;
```

Fortran:

```
INTEGER*4 DOSJIT(FACTOR)
REAL*8 FACTOR
```

DESCRIPTION

DoSampleJitter creates a jitter sampling camera attribute object for antialiasing.

Jitter sampling means that pixels are not sampled at a fixed point, e.g. the pixel center, but at a random point within the pixel. Jitter sampling can also be done in conjunction with supersampling.

The parameter *factor* is a number specifying the percent of a pixel from the fixed point which is the maximum random deviation allowed. Typically *factor* will be between *0.0* and *1.0*. Adjusting this jitter factor gives control of the tradeoff between decreased aliasing and increased noise.

Jitter sampling is enabled/disabled with the *DoSampleJitterSwitch* <*DOSJSW*> attribute.

DEFAULTS

The default *DoSampleJitter* is 0.5.

SEE ALSO

DoSampleJitterSwitch(3D), DoSampleSuper(3D)

NAME

DoSampleJitterSwitch – Create a jitter sampling switch camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleJitterSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOSJSW(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoSampleJitterSwitch creates a jitter sampling switch camera attribute object for antialiasing.

Jitter sampling means that pixels are not sampled at a fixed point, e.g. the pixel center, but at a random point within the pixel. Jitter sampling can also be done in conjunction with supersampling.

The parameter *switchvalue* specifies whether subsequent cameras can jitter sample the scene within each pixel.

DcOn <DCON>
Jitter sampling is enabled.

DcOff <DCOFF>
Each pixel will be sampled at a fixed point.

DEFAULTS

The default *DoSampleJitterSwitch* is *DcOff*.

SEE ALSO

DoSampleJitter(3D), DoSampleSuper(3D)

NAME

DoSampleSuper – Create a supersampling camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleSuper(xsamples, ysamples)
DtInt xsamples;
DtInt ysamples;
```

Fortran:

```
INTEGER*4 DOSSPR(XSAMP, YSAMP)
INTEGER*4 XSAMP
INTEGER*4 YSAMP
```

DESCRIPTION

DoSampleSuper creates a supersampling camera attribute object for antialiasing.

Supersampling means that each pixel of an image is subdivided into subpixels, and a shade is computed for each of the subpixels. A filter is then applied to the shades of the subpixels to obtain the final pixel value.

The parameters *xsamples* and *ysamples* specify how to divide each pixel into subpixels. For example, *DoSampleSuper(2, 3)* would mean sample the pixel twice in the x direction, and three times in the y direction, resulting in 6 subpixels.

Pixels will not be subdivided unless the supersampling switch is turned on.

DEFAULTS

The default *DoSampleSuper* is (2, 2).

SEE ALSO

DoSampleSuperSwitch(3D), DoSampleFilter(3D)

NAME

DoSampleSuperSwitch – Create a supersampling switch camera attribute object for antialiasing

SYNOPSIS

C:

```
DtObject DoSampleSuperSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOSSSW(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoSampleSuperSwitch creates a supersampling switch camera attribute object for antialiasing.

Supersampling means that each pixel of an image is subdivided into subpixels, and a shade is computed for each of the subpixels. A filter is then applied to the shades of the subpixels to obtain the pixel value.

The parameter *switchvalue* specifies whether subsequent cameras can do supersampling.

DcOn <DCON>
: Supersampling is enabled. The number of subpixels per pixel is determined by the *DoSampleSuper <DOSSPR>* attribute.

DcOff <DCOFF>
: The pixels will not be divided into subpixels.

DEFAULTS

The default *DoSampleSuperSwitch* is *DcOff*.

SEE ALSO

DoSampleSuper(3D), DoSampleFilter(3D)

NAME

DoScale – Create a scale geometric tranformation object

SYNOPSIS

C:

```
DtObject DoScale(x,y,z)
DtReal x, y, z;
```

Fortran:

```
INTEGER*4 DOSC(X, Y, Z)
REAL*8 X, Y, Z
```

DESCRIPTION

DoScale creates a scale geometric transformation object that modifies the current transformation matrix attribute resulting in scaling in the X-, Y-, and Z-directions relative to the current origin. The parameters *x*, *y*, and *z* may be any positive or negative value; however, scaling by 0 is undefined.

SEE ALSO

DoLookAtFrom(3D), DoScale(3D), DoShear(3D), DoTransformMatrix(3D), DoTranslate(3D)

NAME

DoShadeIndex – Create a shade index primitive attribute object

SYNOPSIS

C:

```
DtObject DoShadeIndex(index)
DtInt index;
```

Fortran:

```
INTEGER*4 DOSI(INDEX)
INTEGER*4 INDEX
```

DESCRIPTION

DoShadeIndex creates a shade index primitive attribute object. The parameter *index* specifies in which shade range subsequent primitive objects should be rendered when displayed on a pseudocolor device whose shademode is set to *DcRange <DCRNG>*. See *DdSetShadeMode*.

This function accesses the shade range table of a pseudocolor device. Each shade range specifies the minimum and maximum entries in the device's lookup table defining one series of shades of a particular color. The entries in the range increase in intensity from the first shade in the range to the last. Colors produced by the renderer are converted to an intensity between 0.0 and 1.0. This intensity and the current shade index are used to interpolate between the values in the corresponding shade range where an intensity of 0.0 corresponds to the first entry in the shade range and an intensity of 1.0 to the last entry.

DEFAULTS

The default value for *DoShadeIndex* is 0.

SEE ALSO

DdInqShadeMode(3D), DdInqShadeRange(3D), DdSetShadeMode(3D), DdSetShadeRange(3D)

NAME

DoShadowSwitch – Create a shadow switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoShadowSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOSHAS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoShadowSwitch creates a shadow primitive attribute object. The *switchvalue* parameter specifies whether a primitive object will render itself with shadows. Possible values are:

DcOff <*DCOFF*>
: The object will render itself without shadows regardless of the settings of the lights.

DcOn <*DCON*>
: The object will render itself with any shadows cast on it whenever possible.

DEFAULTS

The default value for *DoShadowSwitch* is *DcOff* <*DCOFF*>.

NAME

DoShear – Create a shear geometric transformation object

SYNOPSIS

C:

```
DtObject DoShear(plane, firstdirectionshearvalue,
                                    seconddirectionshearvalue)
DtMajorPlane plane;
DtReal firstdirectionshearvalue, seconddirectionshearvalue;
```

Fortran:

```
INTEGER*4 DOSHR(PLANE, D1SHRV, D2SHRV)
INTEGER*4 PLANE
REAL*8 D1SHRV, D2SHRV
```

DESCRIPTION

DoShear creates a shear geometric transformation object that modifies the current transformation matrix attribute. It results in a directional shear about a major plane specified by the parameter *plane*. Possible values for *plane* are:

DcXY <DCXY>
Z = 0.0

DcYZ <DCYZ>
X = 0.0

DcXZ <DCXZ>
Y = 0.0

The parameter *first_direction_shear_value* specifies the shearing in the first direction of *plane*, and the parameter *second_direction_shear_value* specifies the shearing the second directions of *plane*. For example, let *plane* be *DcXY <DCXY>*. Then *first_direction_shear_value* is the X-direction and *second_direction_shear_value* is the Y-direction.

SEE ALSO

DoLookAtFrom(3D), DoScale(3D), DoRotate(3D), DoTransformMatrix(3D), DoTranslate(3D)

NAME

DoSimplePolygon – Create a simple polygon primitive object

SYNOPSIS

C:

```
DtObject DoSimplePolygon(colormodel, vertextype, vertexcount, vertices, shape)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt vertexcount;
DtReal vertices[];
DtShapeType shape;
```

Fortran:

```
INTEGER*4 DOSPGN(COLMOD, VTXTYP, VTXCNT, VTXS, SHAPE)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 VTXCNT
REAL*8 VTXS(*)
INTEGER*4 SHAPE
```

DESCRIPTION

DoSimplePolygon creates a primitive object that defines a simple polygon. A simple polygon is a planar collection of vertices, forming a single connected contour. The contour may be either convex or concave and may contain any number of vertices. The boundaries of the contour may self-intersect.

The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *vertices* is an array of vertex data. The parameter *vertexcount* specifies the number of vertices in the simple polygon. The parameter *vertices* is an array containing the vertices in order.

The last vertex is automatically connected to the first.

The parameter *shape* hints at the geometry of the simple polygon. Correctly specifying the shape of the contour will result in optimal decomposition when necessary. Incorrectly specifying the shape may result in an incorrect or aborted decomposition. The possible values for *shape* are:

DcConvex <*DCCNVX*>
: Path is wholly convex.

DcConcave <*DCCNCV*>
: Path may be concave but not self-intersecting.

DcComplex <*DCCPLX*>
: Path may be self-intersecting.

Geometric normals to the polygons in the mesh are calculated using the right-hand rule (the fingers of the right hand point in the order of vertices and the thumb points in the direction of the normal). These normals are used to determine the way that the polygons face for backface culling and the shading when the polygons are faceted (no vertex normals provided and smoothflag off) or facet shading has been specified.

ERRORS

DoSimplePolygon must be called with a valid specification for *shape*.

[WARNING - invalid parameter]

DoSimplePolygon must be called with at least three vertices.

[WARNING - insufficient information]

SEE ALSO

DoPolygon(3D), DoPolygonMesh(3D), DoSimplePolygonMesh(3D), VertexTypes(3D)

NAME

DoSimplePolygonMesh – Create a simple polygon mesh primitive object

SYNOPSIS

C:

```
DtObject DoSimplePolygonMesh(colormodel, vertextype, vertexcount, vertices,
                             polygoncount, contours, vertexlist, shape, smoothflag)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt vertexcount;
DtReal vertices[];
DtInt polygoncount;
DtInt contours[];
DtInt vertexlist[];
DtShapeType shape;
DtFlag smoothflag;
```

Fortran:

```
INTEGER*4 DOSPM(COLMOD, VTXTYP, VTXCNT, VTXS,
                PLYCNT, CONTURS, VTXLST, SHAPE, SMOOFL)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 VTXCNT
REAL*8 VTXS(*)
INTEGER*4 PLYCNT
INTEGER*4 CONTURS(PLYCNT)
INTEGER*4 VTXLST(*)
INTEGER*4 SHAPE
INTEGER*4 SMOOFL
```

DESCRIPTION

DoSimplePolygonMesh creates a primitive object that defines a collection of simple polygons, normally interconnected. Each simple polygon must be as described in *DoSimplePolygon*. A *DoPolygonMesh* primitive object may be used for more general polygons consisting of multiple contours.

The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *vertices* is a one-dimensional array of vertex data. The parameter *vertexcount* specifies the number of vertices contained in the *vertices* array.

The parameter *polygoncount* specifies the number of polygons in the mesh. Simple polygons are each composed of one contour. The parameter *contours* is an array of *polygoncount* integers, each of which specifies the number of vertices in the associated polygon.

The parameter *vertexlist* is an array of integers that index the *vertices* array. The first polygon in the mesh begins with the vertex specified by *vertexlist[0]*, and continues using the vertices specified by the following entries in *vertexlist*. The last vertex of a contour is implicitly connected to its first vertex. The next polygon begins with the vertex specified next in the *vertexlist* array.

The parameter *shape* hints at the geometry of the polygons in the mesh. For these polygons, correctly specifying the shape of the polygon will result in optimal polygon decomposition when necessary. Incorrectly specifying the shape may result in an incorrect or aborted decomposition. The possible values for *shape* are:

DcConvex <DCCNVX>
Path is wholly convex.

DcConcave <DCCNCV>
Path may be concave but not self-intersecting.

DcComplex <DCCPLX>
Path may be self-intersecting.

The parameter *smoothflag* specifies that, if vertex normals are not provided, the average normal of the geometric normals from a vertex's adjacent surfaces constitute the normal for shading. (This assumes the simple polygon mesh defines a smooth surface.)

ERRORS

DoSimplePolygonMesh must be called with a valid specification for *shape*.

[WARNING - invalid parameter]

DoSimplePolygonMesh must be called with at least three vertices.

[WARNING - insufficient information]

DoSimplePolygonMesh will issue a warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

SEE ALSO

DoPolygon(3D), DoPolygonMesh(3D), DoSimplePolygon(3D), VertexTypes(3D)

NAME

DoSpecularColor – Create a specular color primitive attribute object

SYNOPSIS

C:

```
DtObject DoSpecularColor(colormodel, color)
DtColorModel colormodel;
DtReal color[ ];
```

Fortran:

```
INTEGER*4 DOSPCC(COLMOD, COLOR)
INTEGER*4 COLMOD
REAL*8 COLOR(*)
```

DESCRIPTION

DoSpecularColor creates a specular color primitive attribute object. The parameter *colormodel* specifies the color model used. The parameter *color* specifies the specular response of a surface to light from directed light sources in the environment.

Specular reflection is the response of a surface to directed incident light where the reflected (specular) light is scattered primarily in the mirror direction. In effect, the specular chromatic response of a surface can be thought of as the color of its shiny highlights.

The specular color is only one of four aspects of the specular component of a surface. The other components are the specular color's intensity, the scattering of the specular reflection, and the applicability of specular components to a surface.

DEFAULTS

The default *DoSpecularColor* is (*DcRGB*, (1.0, 1.0, 1.0)).

SEE ALSO

DoSpecularIntens(3D), DoSpecularFactor(3D), DoSpecularSwitch(3D)

NAME

DoSpecularFactor – Create a specular factor primitive attribute object

SYNOPSIS

C:

```
DtObject DoSpecularFactor(factor)
DtReal factor;
```

Fortran:

```
INTEGER*4 DOSPCF(FACTOR)
REAL*8 FACTOR
```

DESCRIPTION

DoSpecularFactor creates a specular factor primitive attribute object.

Specular reflection is the response of a surface to incident light where the reflected (specular) light is scattered primarily in the mirror direction. The specular factor determines the shape and size of the specular highlight. The larger the value of the specular factor attribute, the more sharp and precise the highlight; the lower the value, the more fuzzy and less precise the highlight.

The parameter *factor* is an exponent in the shading equation Doré uses to calculate specular reflectance in a scene.

The specular factor is only one of four aspects of the specular component of a surface. The other components are the specular color, the specular color's intensity, and the applicability of specular components to a surface.

DEFAULTS

The default *DoSpecularFactor* is 120.

SEE ALSO

DoSpecularColor(3D), DoSpecularIntens(3D), DoSpecularSwitch(3D)

NAME

DoSpecularIntens – Create a specular intensity primitive attribute object

SYNOPSIS

C:

```
DtObject DoSpecularIntens(intensity)
DtReal intensity;
```

Fortran:

```
INTEGER*4 DOSPCI(INTENS)
REAL*8 INTENS
```

DESCRIPTION

DoSpecularIntens creates a specular intensity primitive attribute object. The parameter *intensity* specifies the intensity of the specular response of a surface to light from non-ambient light sources in the environment.

The specular intensity ranges from 0.0 to 1.0, signifying the percentage contribution of specular color to the overall shade of the object's surface.

The specular color is only one of four aspects of the specular component of a surface. The other components are the specular color, the scattering of the specular reflection, and the applicability of specular components to a surface.

DEFAULTS

The default *DoSpecularIntens* is 1.0.

SEE ALSO

DoSpecularColor(3D), DoSpecularFactor(3D), DoSpecularSwitch(3D)

NAME

DoSpecularSwitch – Create a specular switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoSpecularSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOSPCS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoSpecularSwitch creates a specular primitive attribute object. The parameter *switchvalue* specifies whether the surfaces of subsequent primitive objects have a specular component. Possible values are:

DcOff <DCOFF>
The object will render itself without specular shading regardless of the settings for specular color, specular intensity, and specular factor.

DcOn <DCON>
The object will render itself using the specular attributes whenever possible.

The specular switch is only one of four aspects of the specular component of a surface. The other components are the specular color, the specular intensity, and the scattering of the specular reflection.

DEFAULTS

The default for *DoSpecularSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoSpecularFactor(3D), DoSpecularIntens(3D), DoSpecularColor(3D)

NAME

DoStereo – Create a stereo studio attribute object

SYNOPSIS

C:

```
DtObject DoStereo(eyeseparation,distance)
DtReal eyeseparation;
DtReal distance;
```

Fortran:

```
INTEGER*4 DOSTER(EYESEP,DISTNC)
REAL*8 EYESEP
REAL*8 DISTNC
```

DESCRIPTION

DoStereo returns a stereo studio attribute. The attribute requires two input values to specify camera positioning for stereo. The *eyeseparation* parameter is used to specify the distance from the normal camera position to each of the two stereo camera positions (one for each eye). The total distance between the two camera viewpoints will be 2**eyeseparation*. The *distance* parameter specifies the distance from the original camera position to the point along the original camera direction vector at which both stereo camera directions are centered. This controls the "focus" point and becomes important when the distance of the objects being displayed is small enough that it approaches the eye separation distance, that is, when objects become "close". Both parameters are specified in world coordinate space.

DEFAULTS

The default *eyeseparation* is 1.0. The default *distance* is 10.0.

SEE ALSO

DoStereoSwitch(3D)

NAME

DoStereoSwitch – Create a stereo switch studio attribute object

SYNOPSIS

C:

```
DtObject DoStereoSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOSTES(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoStereoSwitch returns a stereo switch studio attribute. The *switchvalue* parameter is used to specify whether or not subsequent cameras are stereo cameras. When the stereo switch is enabled, the *DoStereo* <*DOSTER*> function is used to specify proper stereo camera positioning.

DEFAULTS

The default value for *DoStereoSwitch* is *DcOff* <*DCOFF*>.

SEE ALSO

DoStereo(3D)

NAME

DoSubDivSpec – Create a subdivision specification primitive attribute object

SYNOPSIS

C:

```
DtObject DoSubDivSpec(type,parms)
DtInt type;
DtReal parms[];
```

Fortran:

```
INTEGER*4 DOSDS(TYPE,PARMS)
INTEGER*4 TYPE
REAL*8 PARMS(*)
```

DESCRIPTION

DoSubDivSpec creates a subdivision specification primitive attribute object. *DoSubDivSpec* specifies how to subdivide subsequent analytic primitive objects.

Some primitive objects, such as spheres, cylinders, toruses and patches, are *analytic objects*. That is, they are described by an equation rather than a set of data.

For rendering, these objects must sometimes be decomposed into points, lines, or triangles that are *approximations* of the original analytic surfaces. The finer the decomposition of these surfaces, the closer they approximate the actual object.

The argument *type* can be one of the following:

DcSubDivFixed <DCSDFX>
: For this type, the number of segments along an edge will be equal to 2 to the power of (*parms[0]-1*).

DcSubDivSegments <DCSDSG>
: For this type, the number of segments along an edge will be equal to *parms[0]* whenever possible. When this is not possible, the number of segments along an edge will be as close to *parms[0]* as possible, depending on the primitive being subdivided.

DcSubDivAbsolute <DCSDAB>
: For this type, *parms[0]* equals the maximum deviation allowed anywhere between the curved surface and the nearest plane in the planar approximation of the object.

DcSubDivRelative <DCSDRL>
: For this type, *parms[0]* equals the ratio of the deviation to the length of the side of the approximating polygon. In general, specifying a deviation between 1.5 percent and 4 percent for relative subdivision will result in a reasonable representation of many objects.

DEFAULTS

The default values for *DoSubDivSpec* are *DcSubDivRelative* <DCSDRL> with a *parms[0]* value of 0.1.

NAME

DoSurfaceShade – Create a surface shading primitive attribute object

SYNOPSIS

C:

```
DtObject DoSurfaceShade(object)
DtObject object;
```

Fortran:

```
INTEGER*4 DOSRFS(OBJ)
INTEGER*4 OBJ
```

DESCRIPTION

DoSurfaceShade allows the shading model to be selected on an object-by-object basis. The call takes a single parameter, which is the object handle of the shader which you wish to apply to subsequent primitive objects.

The two shaders provided for *DcRealTime* <*DCRLTM*> rendering are *DcShaderConstant* <*DCSHCN*>, which merely shades objects with their diffuse colors, and *DcShaderLightSource* <*DCSHLS*>, which shades using light sources. The *DcShaderConstant* <*DCSHCN*> shader is slightly faster in some cases, and provides a simple "draw" mode as found in packages like GKS and PHIGS .

NAME

DoText – Create a text primitive object

SYNOPSIS

C:

```
DtObject DoText(position, u, v, string)
DtPoint3 position;
DtVector3 u, v;
DtPtr string;
```

Fortran:

```
INTEGER*4 DOTXT(POSITN, U, V, STRING, N)
REAL*8 POSITN(3)
REAL*8 U(3), V(3)
INTEGER*4 N
CHARACTER*N STRING
```

DESCRIPTION

DoText creates a text primitive object that defines a string of text to be rendered at a specified position and in a specified text plane.

The parameter *position* specifies a point in space where the text will be based. The parameters *u* and *v* are vectors that define two directions. The geometric normal is computed using the right-hand rule such that, while the fingers of the right hand curl in the direction from *v* to the current origin to *u*, the thumb points in the direction of the normal. This normal is used for shading and backface culling. The parameter *string* is the actual text.

FORTRAN SPECIFIC

The parameter *STRING* is string of *N* characters.

SEE ALSO

DoAnnoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextAlign – Create a text alignment primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextAlign(halign, valign)
DtTextAlignHorizontal halign;
DtTextAlignVertical valign;
```

Fortran:

```
INTEGER*4 DOTA(HALIGN, VALIGN)
INTEGER*4 HALIGN
INTEGER*4 VALIGN
```

DESCRIPTION

DoTextAlign creates a text alignment primitive attribute object. The parameters specify the position of the text rectangle relative to the reference point specified in subsequently executed text primitive objects. The alignment depends on the value set by *DoTextPath* <DOTPA> to determine the direction new characters are placed relative to the current text position. See *DoTextPath.*

The parameter *halign* specifies the horizontal position of the text rectangle relative to the reference point. Possible values are:

DcTextHAlignLeft <DCTHAL>
: The left side of the text rectangle passes through the text position.

DcTextHAlignCenter <DCTHAC>
: The text position lies midway between the left and right sides of the text rectangle.

DcTextHAlignRight <DCTHAR>
: The right side of the text rectangle passes through the text position.

DcTextHAlignNormal <DCTHAN>
: The most natural alignment is achieved by selecting from the other alignment values based on the value of the current text path attribute.

The parameter *valign* specifies the vertical position of the text rectangle relative to the reference point. Possible values are:

DcTextVAlignTop <DCTVAT>
: The top of the text rectangle passes through the text position.

DcTextVAlignCap <DCTVAC>
: If the text path is *DcTextPathLeft* <DCTPL> or *DcTextPathRight* <DCTPR>, the text position lies on the capline of the whole string. If the text path is *DcTextPathUp* <DCTPU> or *DcTextPathDown* <DCTPD>, the text position lies on the capline of the topmost character.

DcTextVAlignHalf <DCTVAH>
: If the text path is *DcTextPathLeft* <DCTPL> or *DcTextPathRight* <DCTPR>, the text position lies on the halfline of the whole string. If the text path is *DcTextPathUp* <DCTPU> or *DcTextPathDown* <DCTPD>, the text position lies on a line halfway between the halflines of the top and bottom characters.

DcTextVAlignBase <DCTVAB>
: If the text path is *DcTextPathLeft* <DCTPL> or *DcTextPathRight* <DCTPR>, the text position lies on the baseline of the whole string. If the text path is *DcTextPathUp* <DCTPU> or *DcTextPathDown* <DCTPD>, the text position lies on the baseline of the bottom character in the string.

DcTextVAlignBottom <DCTVAM>
The bottom of the text rectangle passes through the text position.

DcTextVAlignNormal <DCTVAN>
The most natural alignment is achieved by selecting from the other alignment values based on the value of the current text path attribute.

DEFAULTS

The default *DoTextAlign* is (*DcTextHAlignNormal <DCTHAN>*, *DcTextVAlignNormal <DCTVAN>*).

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextExpFactor – Create a text expansion factor primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextExpFactor(textexp)
DtReal textexp;
```

Fortran:

```
INTEGER*4 DOTEF(TXEXP)
REAL*8 TXEXP
```

DESCRIPTION

DoTextExpFactor creates a text expansion factor primitive attribute object. The parameter *textexp* specifies a scale factor for the character width such that the character aspect ratio deviates from that indicated by the font designer by *textexp*. Possible values are:

Negative numbers
: The character width contracts by the specified amount.

Positive numbers
: The character width expands by the specified amount.

DEFAULTS

The default *DoTextExpFactor* is 1.0.

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextFont – Create a text font primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextFont(font)
DtFont font;
```

Fortran:

```
INTEGER*4 DOTF(FONT)
INTEGER*4 FONT
```

DESCRIPTION

DoTextFont creates a text font primitive attribute object. The parameter *font* defines the font to be used for subsequently executed text primitive objects. Possible fonts are listed in the following table:

C Font Name	Fortran Font Name
DcAstrology	DCFAST
DcComplexCyrillic	DCFCC
DcComplexGreek	DCFCG
DcComplexSmallGreek	DCFCSG
DcComplexSmallItalic	DCFCSI
DcComplexSmallRoman	DCFCSR
DcComplexItalic	DCFCI
DcComplexRoman	DCFCR
DcComplexScript	DCFCS
DcDuplexRoman	DCFDR
DcLowerCaseMathematics	DCFLCM
DcGothicGerman	DCFGG
DcGothicEnglish	DCFGE
DcGothicItalian	DCFGI
DcHelvetica (polygonal font)	DCFHLV
DcMusic	DCFMUS
DcMeteorology	DCFMUS
DcPlainGreek	DCFPG
DcPlainRoman	DCFPR
DcSimplexRoman	DCFSR
DcSimplexGreek	DCFSG
DcSimplexScript	DCFSS
DcSymbols	DCFSYM
DcTriplexItalic	DCFTI
DcTriplexRoman	DCFTR
DcUpperCaseMathematics	DCFUCM

DEFAULTS

The default *DoTextFont* is *DcPlainRoman* <*DCFPR*>.

ERRORS

DoTextFont will fail if an invalid *font* is specified.

[WARNING - value out of range]

DoTextFont will fail if memory cannot be allocated for character storage.

[SEVERE - unable to allocate memory]

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextHeight – Create a text height primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextHeight(textheight)
DtReal textheight;
```

Fortran:

```
INTEGER*4 DOTH(TXHGHT)
REAL*8 TXHGHT
```

DESCRIPTION

DoTextHeight creates a text height primitive attribute object. The parameter *textheight* specifies the nominal height of a capital letter for subsequently executed text primitive objects. The *textheight* parameter should be a positive value.

DEFAULTS

The default *DoTextHeight* is 1.0.

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextPath – Create a text path primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextPath(textpath)
DtTextPath textpath;
```

Fortran:

```
INTEGER*4 DOTPA(TXPATH)
INTEGER*4 TXPATH
```

DESCRIPTION

DoTextPath creates a text path primitive attribute object. The parameter *textpath* specifies the drawing direction for subsequently executed text primitive objects. Possible values are:

DcTextPathRight <DCTPR>
Each succeeding character is to the right of the previous character.

DcTextPathLeft <DCTPL>
Each succeeding character is to the left of the previous character.

DcTextPathUp <DCTPU>
Each succeeding character is above the previous character.

DcTextPathDown <DCTPD>
Each succeeding character is below the previous character.

DoTextUpVector is used to alter the orientation of the characters themselves.

DEFAULTS

The default *DoTextPath* is *DcTextPathRight <DCTPR>*.

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPrecision(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextPrecision – Create a text precision primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextPrecision(precision)
DtTextPrecision precision;
```

Fortran:

```
INTEGER*4 DOTPR(PRECIS)
INTEGER*4 PRECIS
```

DESCRIPTION

DoTextPrecision creates a text precision primitive attribute object. The parameter *precision* specifies the accuracy with which text is drawn for subsequently executed text primitive objects. Possible values are:

DcStringPrecision <DCSTRP>
The attributes used to render text objects are text font, text height, and text expansion factor. The text is clipped in a device-dependent manner.

DcCharacterPrecision <DCCHRP>
The attributes used to render text objects are text font, text height, text expansion factor, text up vector, text space, text path and text alignment. The text is clipped, at the least, on a character-by-character basis.

DcStrokePrecision <DCSTKP>
All text attributes are applied exactly and, if applicable, lighting and shading calculations are also performed on the text primitive object.

DEFAULTS

The default *DoTextPrecision* is *DcStrokePrecision <DCSTKP>*.

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextSpace(3D), DoTextUpVector(3D)

NAME

DoTextSpace – Create a text space primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextSpace(textspace)
DtReal textspace;
```

Fortran:

```
INTEGER*4 DOTSP(TXSPAC)
REAL*8 TXSPAC
```

DESCRIPTION

DoTextSpace creates a text space primitive attribute object. The parameter *textspace* specifies the amount of additional space to insert between adjacent characters in subsequently executed text primitive objects.

The *textspace* parameter is specified as a scaling factor (fraction) of the font-nominal character width to be used as additional spacing. The specified spacing can be positive, negative or 0. When this scaling factor is negative, characters overlap each other by the specified amount.

DEFAULTS

The default *DoTextSpace* is 0.0.

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextUpVector(3D)

NAME

DoTextUpVector – Create a text up vector primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextUpVector(xup, yup)
DtReal xup;
DtReal yup;
```

Fortran:

```
INTEGER*4 DOTUV(XUP, YUP)
REAL*8 XUP
REAL*8 YUP
```

DESCRIPTION

DoTextUpVector creates a text up vector primitive attribute object. The parameter *xup* specifies the planar shearing or slant of the characters on the text plane in the X-direction for subsequently executed text primitive objects. The parameter *yup* specifies the planar shearing or slant in the Y-direction.

The text up vector determines the up direction for the top of each character. The parameters can be treated as a single, non-normalized, two-dimensional vector.

DEFAULTS

The default *DoTextUpVector* is (0.0, 1.0).

SEE ALSO

DoAnnoText(3D), DoText(3D), DoTextAlign(3D), DoTextExpFactor(3D), DoTextFont(3D), DoTextHeight(3D), DoTextPath(3D), DoTextPrecision(3D), DoTextSpace(3D)

NAME

DoTextureAntiAlias – Create a texture attribute object that controls antialiasing of textures

SYNOPSIS

C:

```
DtObject DoTextureAntiAlias(mode)
DtTextureAntiAliasMode mode;
```

Fortran:

```
INTEGER*4 DOTAA(MODE)
INTEGER*4 MODE
```

DESCRIPTION

DoTextureAntiAlias creates a texture attribute object that controls the antialiasing of textures. The parameter *mode* specifies the type of antialiasing to be done. The possible values of *mode* are:

DcTextureAntiAliasNone <DCTAAN>
No antialiasing of the texture is to be performed.

DcTextureAntiAliasMIP <DCTAAM>
MIP mapping is to be used to antialias the texture.

DcTextureAntiAliasAdaptive <DCTAAA>
Adaptive techniques are to be used to antialias the texture.

DEFAULTS

The default value for *DoTextureAntiAlias* is *DcTextureAntiAliasNone <DCTAAN>*.

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D)

NAME

DoTextureExtendUV – Create a texture attribute object that controls texturing behavior beyond the boundary of a 2-dimensional texture

SYNOPSIS

C:

```
DtObject DoTextureExtendUV(umode, vmode)
DtExtendMode umode;
DtExtendMode vmode;
```

Fortran:

```
INTEGER*4 DOTXUV(UMODE, VMODE)
INTEGER*4 UMODE
INTEGER*4 VMODE
```

DESCRIPTION

DoTextureExtendUV creates a texture attribute object that specifies what happens to the texture value when the **u** or **v** coordinates of the primitive object extends beyond the limits of a 2-dimensional texture. The parameter *umode* specifies the extension mode in the **u** direction and *vmode* specifies the extension mode in the **v** direction. The possible values of *umode* and *vmode* are:

DcExtendNone <DCXNON>
: If the coordinate is outside the range [0..1] then no texture mapping is done. It is as if no texture map had been specified.

DcExtendBlack <DCXBLK>
: If the coordinate is outside the range [0..1] then the value computed by the texture will be zero for every channel of data.

DcExtendClamp <DCXCLP>
: If the coordinate is outside the range [0..1] then the value computed by the texture is the value at the nearest edge. For example, if **u** is 3.5 then **u** is clamped to 1.0, thus the edge values are extended.

DcExtendRepeat <DCXRP>
: If the coordinate is outside the range [0..1] then the texture is repeated. Effectively, if the coordinate is **u** then the texture coordinate used for mapping would be (**u** - floor(**u**)).

DEFAULTS

The default value for *DcTextureExtendUV* is *DcExtendNone* <DCXNON> for both modes.

SEE ALSO

DoTextureExtendUVW(3D), DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D)

NAME

DoTextureExtendUVW – Create a texture attribute object that controls texturing behavior beyond the boundary of a 3-dimensional texture

SYNOPSIS

C:

```
DtObject DoTextureExtendUVW(umode, vmode, wmode)
DtExtendMode umode;
DtExtendMode vmode;
DtExtendMode wmode;
```

Fortran:

```
INTEGER*4 DOTXW(UMODE, VMODE, WMODE)
INTEGER*4 UMODE
INTEGER*4 VMODE
INTEGER*4 WMODE
```

DESCRIPTION

DoTextureExtendUVW creates a texture attribute object that specifies what happens to the texture value when the **u**, **v** or **w** coordinates of the primitive object extends beyond the limits of a 3-dimensionsal texture. The parameter *umode* specifies the extension mode in the **u** direction, *vmode* specifies the extension mode in the **v** direction, and *wmode* specifies the extension mode in the **w** direction. The possible values of *umode, vmode* and *wmode* are:

DcExtendNone <DCXNON>
: If the coordinate is outside the range [0..1] then no texture mapping is done. It is as if no texture map had been specified.

DcExtendBlack <DCXBLK>
: If the coordinate is outside the range [0..1] then the value computed by the texture will be zero for every channel of data.

DcExtendClamp <DCXCLP>
: If the coordinate is outside the range [0..1] then the value computed by the texture is the value at the nearest edge. For example, if **u** is 3.5 then **u** is clamped to 1.0, thus the edge values are extended.

DcExtendRepeat <DCXRP>
: If the coordinate is outside the range [0..1] then the texture is repeated. Effectively, if the coordinate is **u** then the texture coordinate used for mapping would be (**u** - floor(**u**)).

DEFAULTS

The default value for *DcTextureExtendUVW* is *DcExtendNone <DCXNON>* for all three modes.

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureMapExtendUV(3D)

NAME

DoTextureMapBump – Create a bump map primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextureMapBump(operator, mapping, raster)
DtMapOperator operator;
DtObject mapping;
DtObject raster;
```

Fortran:

```
INTEGER*4 DOTMB(OPRATR, MAPPNG, RASTER)
INTEGER*4 OPRATR
INTEGER*4 MAPPNG
INTEGER*4 RASTER
```

DESCRIPTION

DoTextureMapBump creates a primitive attribute object that specifies a bump map, i.e. a texture map for perturbing shading normals.

A bump-mapped primitive object can be affected by more than one *DoTextureMapBump* attribute. The parameter *operator* specifies whether subsequent primitive objects will be affected by this bump map only, or also by other bump maps higher up in the scene hierarchy. The possible values for *operator* are:

DcMapReplace <DCMR>
: Any bump maps already defined will be ignored by subsequent primitive objects.

DcMapAdd <DCMADD>
: Use this map in addition to the bump maps higher up in the hierarchy. In case of overlapping textures, the one that is defined closer to the primitive object affected by the textures will be evaluated first. Note that a primitive object can only be affected by textures above it in the hierarchy.

The parameter *mapping* specifies which mapping procedure to use. Possible values for *mapping* are:

DcStdBumpMap <DCSBM>
: Use the default bump mapping method the renderer provides.

The parameter *raster* is the actual bump map data to be used by the texturing procedure. *raster* is a handle to an object created with either *DoRaster <DORS>* or *DoFileRaster <DOFRS>*. One channel of data is typically needed for bump maps. If the raster data is in the standard Doré format, the following convention is used. For rasters of type *DcRasterZ <DCRZ>*, *DcRasterRGBZ <DCRRZ>* and *DcRasterRGBAZ <DCRRAZ>* the Z channel is used. For rasters of type *DcRasterA <DCRA>* the alpha channel is used. For rasters of type *DcRasterRGB <DCRRGB>* and *DcRasterRGBA <DCRRA>* a single intensity value is calculated from RGB data for every pixel. This intensity value is then used as bump map data. Note that some renderers may not adhere to this convention.

DoTextureMapBump is a primitive attribute and is inherited by subsequent primitive objects, as are all primitive attributes. Unlike most other primitive attributes though, *DoTextureMapBump* also inherits a set of attributes, i.e. the texture attributes. The texture attributes specify things such as which set of primitive **uv** coordinates to use, how to extend a texture when the primitive **uv** coordinates extend beyond the limits of the texture, how to transform the primitive **uv** coordinates before applying the texture, etc. The current values of the texture attributes are bound to the *DoTextureMap-*

Bump object when it is executed during traversal. Thus the texture attributes are only indirectly bound to the primitive objects.

ERRORS

DoTextureMapBump will fail if *raster* is not a *DoRaster* object or a *DoFileRaster* object.

[WARNING - invalid or deleted object]

SEE ALSO

DoTextureMapBumpSwitch(3D), DoTextureUVIndex(3D), DoTextureUVWIndex(3D), DoTextureScaleUV(3D), DoTextureScaleUVW(3D), DoTextureTranslateUV(3D), DoTextureTranslateUVW(3D), DoTextureExtendUV(3D), DoTextureExtendUVW(3D), DoTextureOp(3D), DoTextureAntiAlias(3D)

NAME

DoTextureMapBumpSwitch – Create a primitive attribute object for enabling/disabling bump mapping

SYNOPSIS

C:

```
DtObject DoTextureMapBumpSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOTMBS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoTextureMapBumpSwitch creates a primitive attribute object that enables/disables bump mapping, i.e. perturbing shading normals via a texture map. The *switchvalue* parameter is used to specify whether or not the shading normal of subsequent primitive objects can be modified by bump mapping.

DcOff <DCOFF>
The primitive object will ignore any bump maps that have been defined with *DoTextureMapBump <DOTMDC>*.

DcOn <DCON>
The primitive object will be bump mapped as specified by the *DoTextureMapBump* attribute, whenever possible.

DEFAULTS

The default for *DoTextureMapBumpSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoTextureMapBump(3D)

NAME

DoTextureMapDiffuseColor – Create a diffuse color texture map primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextureMapDiffuseColor(operator, mapping, raster)
DtMapOperator operator;
DtObject mapping;
DtObject raster;
```

Fortran:

```
INTEGER*4 DOTMDC(OPRATR, MAPPNG, RASTER)
INTEGER*4 OPRATR
INTEGER*4 MAPPNG
INTEGER*4 RASTER
```

DESCRIPTION

DoTextureMapDiffuseColor creates a primitive attribute object that specifies a texture map for diffuse color.

A texture-mapped primitive object can be affected by more than one *DoTextureMapDiffuseColor* attribute. The parameter *operator* specifies whether subsequent primitive objects will be affected by this diffuse color texture only, or also by other diffuse color textures higher up in the scene hierarchy. The possible values for *operator* are:

DcMapReplace <DCMRPL>
: Any diffuse color textures already defined will be ignored by subsequent primitive objects.

DcMapAdd <DCMADD>
: Use this texture in addition to the diffuse color textures higher up in the hierarchy. In case of overlapping textures, the one that is defined closest to the primitive object affected by the textures will be evaluated first. Note that a primitive object can only be affected by textures above it in the hierarchy.

The parameter *mapping* specifies which mapping procedure to use. Possible values for *mapping* are:

DcStdTableLookup <DCSTLU>
: Use the default 2-dimensional static texture mapping method the renderer provides.

DcStd3dTableLookup <DCS3TL>
: Use the default 3-dimensional static texture mapping method the renderer provides.

The parameter *raster* is the actual texture map data to be used by the texturing procedure. *raster* is a handle to an object created with *DoRaster <DORS>*, or *DoFileRaster <DOFRS>*. The number of channels of data used for mapping the diffuse color depends on the colormodel. Three channels are required if an RGB colormodel is used for the diffuse color. If the raster data is in the standard Doré format, the following convention is used. For rasters of type *DcRasterRGB <DCRRGB>*, *DcRasterRGBA <DCRRA>*, *DcRasterRGBZ <DCRRZ>*, and *DcRasterRGBAZ <DCRRAZ>* the RGB channels are used. Some texture maps also use the alpha channel (see *DoTextureOp*). Rasters of type *DcRasterA <DCRA>* and *DcRasterZ <DCRZ>* cannot be used as diffuse color texture maps in conjunction with the RGB colormodel.

DoTextureMapDiffuseColor is a primitive attribute and is inherited by subsequent primitive objects, as are all primitive attributes. Unlike most other primitive attributes though, *DoTextureMapDiffuseColor* also inherits a set of attributes, i.e. the texture attributes. The texture attributes specify things such as which set of primitive **uv** coordinates to use, how to extend a texture when the primitive **uv** coordinates extend beyond the limits of the texture, how to transform the primitive **uv** coordinates before applying the texture, etc. The current values of the texture attributes are bound to the *DoTextureMapDiffuseColor* object when it is executed during traversal. Thus the texture attributes are only indirectly bound to the primitive objects.

ERRORS

DoTextureMapDiffuseColor will fail if *raster* is not a *DoRaster* object or a *DoFileRaster* object.

[WARNING - invalid or deleted object]

SEE ALSO

DoDiffuseColor(3D), DoTextureMapDiffuseColorSwitch(3D), DoTextureUVIndex(3D), DoTextureUVWIndex(3D), DoTextureScaleUV(3D), DoTextureScaleUVW(3D), DoTextureTranslateUV(3D), DoTextureTranslateUVW(3D), DoTextureExtendUV(3D), DoTextureExtendUVW(3D), DoTextureOp(3D), DoTextureAntiAlias(3D)

NAME

DoTextureMapDiffuseColorSwitch – Create a primitive attribute object for enabling/disabling texture mapping of diffuse color

SYNOPSIS

C:

```
DtObject DoTextureMapDiffuseColorSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOTMDS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoTextureMapDiffuseColorSwitch creates a primitive attribute object that enables/disables texturing mapping of diffuse color. The *switchvalue* parameter is used to specify whether or not the diffuse color of subsequent objects can be modified by texture mapping.

DcOff <DCOFF>
The primitive object will ignore any diffuse color texture maps that have been defined with *DoTextureMapDiffuseColor <DOTMDC>*.

DcOn <DCON>
The diffuse color will be mapped as specified by the *DoTextureMapDiffuseColor* attribute whenever possible. Note that the *DoDiffuseSwitch* primitive attribute must be also be *DcOn <DCON>*.

DEFAULTS

The default for *DoTextureMapDiffuseColorSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoDiffuseColor(3D), DoDiffuseSwitch(3D), DoTextureMapDiffuseColor(3D)

NAME

DoTextureMapEnviron – Create an environment map primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextureMapEnviron(operator, mapping, raster)
DtMapOperator operator;
DtObject mapping;
DtObject raster;
```

Fortran:

```
INTEGER*4 DOTME(OPRATR, MAPPNG, RASTER)
INTEGER*4 OPRATR
INTEGER*4 MAPPNG
INTEGER*4 RASTER
```

DESCRIPTION

DoTextureMapEnviron creates a primitive attribute object that specifies an environment map.

A primitive object can be affected by more then one *DoTextureMapEnviron* attribute. The parameter *operator* specifies whether subsequent primitive objects will be affected by this environment map only, or also by other environment maps higher up in the scene hierarchy. The possible values for *operator* are:

DcMapReplace <DCMRPL>
: Any environment maps already defined will be ignored by subsequent primitive objects.

DcMapAdd <DCMADD>
: Use this map in addition to the environment map higher up in the hierarchy. In case of overlapping textures, the one that is defined closest to the primitive object affected by the textures will be evaluated first. Note that a primitive object can only be affected by textures above it in the hierarchy.

The parameter *mapping* specifies which mapping procedure to use. Possible values for *mapping* are:

DcStdSphereEnvironMap <DCSEM>
: Use the default spherical environment mapping method the renderer provides.

DcStdCubeEnvironMap <DCSEM>
: Use the default cube environment mapping method the renderer provides.

The parameter *raster* contains the actual environment map data to be used by the texturing procedure. *raster* is a handle to an object created with *DoRaster <DORS>* or *DoFileRaster <DOFRS>*. The number of channels of data used for the environment mapping depends on the colormodel. Three channels are required if the RGB colormodel is used. If the raster data is in the standard Doré format, the following convention is used. For rasters of type *DcRasterRGB <DCRRGB>*, *DcRasterRGBA <DCRRA>*, *DcRasterRGBZ <DCRRZ>* and *DcRasterRGBAZ <DCRRAZ>* the RGB channels are used. Rasters of type *DcRasterA <DCRA>* and *DcRasterZ <DCRZ>* cannot be used as environment maps in conjunction with the RGB colormodel.

DoTextureMapEnviron is a primitive attribute and is inherited by subsequent primitive objects, as are all primitive attributes. Unlike most other primitive attributes though, *DoTextureMapEnviron* also inherits a set of attributes, i.e. the texture attributes. The texture attributes specify things such as which set of primitive **uv** coordinates to use, how to extend a texture when the primitive **uv** coordinates extend beyond the limits of the texture, how to transform the primitive **uv** coordinates be-

fore applying the texture, etc. The current values of the texture attributes are bound to the *DoTextureMapEnviron* object when it is executed during traversal. Thus the texture attributes are only indirectly bound to the primitive objects.

SEE ALSO

DoDiffuseColor(3D), DoTextureMapEnvironSwitch(3D), DoTextureUVIndex(3D), DoTextureUVWIndex(3D), DoTextureScaleUV(3D), DoTextureScaleUVW(3D), DoTextureTranslateUV(3D), DoTextureTranslateUVW(3D), DoTextureExtendUV(3D), DoTextureExtendUVW(3D), DoTextureOp(3D), DoTextureAntiAlias(3D)

NAME

DoTextureMapEnvironSwitch – Create a primitive attribute object for enabling/ disabling environment mapping

SYNOPSIS

C:

```
DtObject DoTextureMapEnvironSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOTMES(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoTextureMapEnvironSwitch creates a primitive attribute object that enables/disables environment mapping. The *switchvalue* parameter is used to specify whether or not the shading of subsequent primitive objects can be affected by environment mapping.

DcOff <*DCOFF*>
: The primitive object will ignore any environment maps that have been defined with *DoTextureMapEnviron* <*DOTMDC*>.

DcOn <*DCON*>
: The primitive object will be environment mapped as specified by the *DoTextureMapEnviron* attribute, whenever possible.

DEFAULTS

The default for *DoTextureMapEnvironSwitch* is *DcOff* <*DCOFF*>.

SEE ALSO

DoTextureMapEnviron(3D)

NAME

DoTextureMapTranspIntens – Create a transparent intensity texture map primitive attribute object

SYNOPSIS

C:

```
DtObject DoTextureMapTranspIntens(operator, mapping, raster)
DtMapOperator operator;
DtObject mapping;
DtObject raster;
```

Fortran:

```
INTEGER*4 DOTMTI(OPRATR, MAPPNG, RASTER)
INTEGER*4 OPRATR
INTEGER*4 MAPPNG
INTEGER*4 RASTER
```

DESCRIPTION

DoTextureMapTranspIntens creates a primitive attribute object that specifies a texture map for transparent intensity.

Texture mapped primitive objects can be affected by more than one *DoTextureMapTranspIntens* attribute. The parameter *operator* specifies whether subsequent primitive objects will be affected by this transparent intensity texture only, or also by other transparent intensity textures higher up in the scene hierarchy. The possible values for *operator* are:

DcMapReplace <DCMRPL>
: Any transparent intensity textures already defined will be ignored by subsequent primitive objects.

DcMapAdd <DCMADD>
: Use this texture in addition to the transparent intensity textures higher up in the hierarchy. In case of overlapping textures, the one that is defined closest to the primitive object affected by the textures will be evaluated first. Note that a primitive object can only be affected by textures above it in the hierarchy.

The parameter *mapping* specifies which mapping procedure to use. Possible values for *mapping* are:

DcStdTableLookup <DCSTLU>
: Use the default 2-dimensional static texture mapping method the renderer provides.

DcStd3dTableLookup <DCS3TL>
: Use the default 3-dimensional static texture mapping method the renderer provides.

The parameter *raster* contains the actual texture map data to be used by the texturing procedure. *raster* is a handle to an object created with *DoRaster <DORS>*, or *DoFileRaster <DOFRS>*. Only one channel of data is (usually) needed for mapping transparent intensity. If the raster data is in the standard Doré format, the following convention is used. For rasters of type *DcRasterA <DCRA>* the alpha channel is used. For rasters of type *DcRasterZ <DCRZ>* the Z channel is used. For rasters of type *DcRasterRGB <DCRRGB>*, *DcRasterRGBA <DCRRA>*, *DcRasterRGBZ <DCRRZ>*, and *DcRasterRGBAZ <DCRRAZ>* a single intensity value is calculated from RGB data for every pixel. This value is then used as the map data. Some texture maps also use the alpha channel (see *DoTextureOp*). Note that some renderers may not adhere to this convention.

DoTextureMapTranspIntens is a primitive attribute and is inherited by subsequent primitive objects, as are all primitive attributes. Unlike most other primitive attributes though, *DoTextureMapTranspIntens* also inherits a set of attributes, i.e. the texture attributes. The texture attributes specify things such as which set of primitive *uv* coordinates to use, how to extend a texture when the primitive *uv* coordinates extend beyond the limits of the texture, whether to apply a transformation to the primitive *uv* coordinates before applying the texture, etc. The current values of the texture attributes are bound to the *DoTextureMapTranspIntens* object when it is executed during traversal. Thus the texture attributes are only indirectly bound to the primitive objects.

ERRORS

DoTextureMapTranspIntens will fail if *raster* is not a *DoRaster* object or a *DoFileRaster* object.

[WARNING - invalid or deleted object]

SEE ALSO

DoTranspIntens(3D), DoTextureMapTranspIntensSwitch(3D), DoTextureUVIndex(3D), DoTextureUVWIndex(3D), DoTextureScaleUV(3D), DoTextureScaleUVW(3D), DoTextureTranslateUV(3D), DoTextureTranslateUVW(3D), DoTextureExtendUV(3D), DoTextureExtendUVW(3D), DoTextureOp(3D), DoTextureAntiAlias(3D)

NAME

DoTextureMapTranspIntensSwitch – Create a primitive attribute object for enabling/disabling texture mapping of transparent intensity

SYNOPSIS

C:

```
DtObject DoTextureMapTranspIntensSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOTMTS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoTextureMapTranspIntensSwitch creates a primitive attribute object that enables/disables texturing mapping of transparent intensity. The *switchvalue* parameter is used to specify whether or not the transparent intensity of subsequent objects can be modified by texture mapping.

DcOff <DCOFF>
The primitive object will ignore any transparent intensity texture maps that have been defined with *DoTextureMapTranspIntens <DOTMDC>*.

DcOn <DCON>
The transparent intensity will be mapped as specified by the *DoTextureMapTranspIntens* attribute whenever possible. Note that the *DoTranspSwitch* primitive attribute must be also be *DcOn <DCON>*.

DEFAULTS

The default for *DoTextureMapTranspIntensSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoTranspIntens(3D), DoTranspIntensSwitch(3D), DoTextureMapTranspIntens(3D)

NAME

DoTextureOp – Create a texture attribute object that controls how the texture value is combined with the current value

SYNOPSIS

C:

```
DtObject DoTextureOp(operator)
DtTextureOperator operator;
```

Fortran:

```
INTEGER*4 DOTOP(OP)
INTEGER*4 OP
```

DESCRIPTION

DoTextureOp creates a texture attribute object that controls how the computed texture value combines with the value already computed by the renderer. Note that another texture may already have affected the value computed by the renderer. The parameter *operator* specifies the combination rule. The possible values of *operator* are:

DcTextureReplace <DCTREP>
The value computed from the texture replaces the value already computed by the renderer.

DcTextureMultiply <DCTMUL>
The value computed from the texture is used to scale the value already computed by the renderer.

DcTextureBlend <DCTBLD>
This is only valid if the texture also includes an alpha channel. The value computed from the texture is combined, using alpha, with the value already computed by the renderer:
new_value = computed_value(alpha-1) + texture_value*alpha*

DcTextureAdd <DCTADD>
The value computed from the texture is added to the value already computed by the renderer.

Note that not all renderers will support all types of operators.

DEFAULTS

The default value for *DoTextureOp* is *DcTextureReplace <DCTREP>*.

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D)

NAME

DoTextureScaleUV – Create a texture attribute object that sets the scale factors for primitive **uv** coordinates

SVNOPSIS

C:

```
DtObject DoTextureScaleUV(su, sv)
DtReal su;
DtReal sv;
```

Fortran:

```
INTEGER*4 DOTSUV(SU, SV)
REAL*8 SU
REAL*8 SV
```

DESCRIPTION

DoTextureScaleUV creates a texture attribute object that defines the scaling values to be applied to the **uv** coordinates of a primitive object before they are used to access a 2-dimensional texture. The parameter *su* is the scale factor in the **u** direction, and *sv* is the scale factor in the **v** direction. *DoTextureScaleUV* is used in conjunction with *DoTextureTranslateUV* <*DOTTUV*>to completely specify how a primitive object's **uv** coordinates are transformed before they are used to access the texture. The *DoTextureScaleUV* is always applied before the *DoTextureTranslateUV*, regardless of the sequence in the scene database.

DEFAULTS

The default value for *DoTextureScaleUV* is (1.,1.).

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureScaleUVW(3D), DoTextureTranslateUV(3D)

NAME

DoTextureScaleUVW – Create a texture attribute object that sets the scale factors for primitive **uvw** coordinates

SVNOPSIS

C:

```
DtObject DoTextureScaleUVW(su, sv, sw)
DtReal su;
DtReal sv;
DtReal sw;
```

Fortran:

```
INTEGER*4 DOTSW(SU, SV, SW)
REAL*8 SU
REAL*8 SV
REAL*8 SW
```

DESCRIPTION

DoTextureScaleUVW creates a texture attribute object that defines the scaling values to be applied to the **uvw** coordinates of a primitive object before they are used to access a 3-dimensionsal texture. The parameter *su* is the scale factor in the **u** direction, *sv* is the scale factor in the **v** direction, and *sw* is the scale factor in the **w** direction. *DoTextureScaleUVW* is used in conjunction with *DoTextureTranslateUVW* <*DOTTW*>to completely specify how a primitive object's **uvw** coordinates are transformed before they are used to access the texture. The *DoTextureScaleUVW* is always applied before the *DoTextureTranslateUVW*, regardless of the sequence in the scene database.

DEFAULTS

The default value for *DoTextureScaleUVW* is (1.,1.,1.).

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureScaleUV(3D), DoTextureTranslateUVW(3D)

NAME

DoTextureTranslateUV – Create a texture attribute object that sets the translate factors for primitive **uv** coordinates

SYNOPSIS

C:

```
DtObject DoTextureTranslateUV(tu, tv)
DtReal tu;
DtReal tv;
```

Fortran:

```
INTEGER*4 DOTTUV(TU, TV)
REAL*8 TU
REAL*8 TV
```

DESCRIPTION

DoTextureTranslateUV creates a texture attribute object that defines the translation values to be applied to the **uv** coordinates for a primitive object before they are used to access a 2-dimensional texture. *tu* is the translation factor in the **u** direction and *tv* is the translation factor in the **v** direction. *DoTextureTranslateUV* is used in conjunction with *DoTextureScaleUV <DOTSUV>* to completely specify how a primitive object's **uv** coordinates are transformed before they are used to access the texture. The *DoTextureScaleUV <DOTSUV>* is always applied before the *DoTextureTranslateUV*, regardless of the sequence in the scene database.

DEFAULTS

The default value for *DoTextureTranslateUV* is (0.,0.).

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureScaleUV(3D), DoTextureTranslateUVW(3D)

NAME

DoTextureTranslateUVW – Create a texture attribute object that sets the translate factors for primitive **uvw** coordinates

SYNOPSIS

C:

```
DtObject DoTextureTranslateUVW(tu, tv, tw)
DtReal tu;
DtReal tv;
DtReal tw;
```

Fortran:

```
INTEGER*4 DOTTW(TU, TV, TW)
REAL*8 TU
REAL*8 TV
REAL*8 TW
```

DESCRIPTION

DoTextureTranslateUVW creates a texture attribute object that defines the translation values to be applied to the **uvw** coordinates for a primitive object before they are used to access a 3-dimensional texture. The parameter *tu* is the translation factor in the **u** direction, *tv* is the translation factor in the **v** direction, and *tw* is the translation factor in the **w** direction. *DoTextureTranslateUVW* is used in conjunction with *DoTextureScaleUVW <DOTSW>* to specify completely how a primitive object's **uvw** coordinates are transformed before they are used to access the texture. The *DoTextureScaleUVW <DOTSW>* is always applied before the *DoTextureTranslateUVW*, regardless of the sequence in the scene database.

DEFAULTS

The default value for *DoTextureTranslateUV* is (0.,0.,0.).

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureScaleUVW(3D), DoTextureTranslateUV(3D)

NAME

DoTextureUVIndex – Create a texture attribute object that specifies which **uv** coordinates of a primitive object to use

SYNOPSIS

C:

```
DtObject DoTextureUVIndex(index)
DtInt index;
```

Fortran:

```
INTEGER*4 DOTUVI(INDEX)
INTEGER*4 INDEX
```

DESCRIPTION

DoTextureUVIndex creates a texture attribute object that specifies which set of **uv** coordinates in a primitive object to use when doing 2-dimensional texture mapping.

Each primitive object can have more then one **uv** coordinate per vertex. The **uv** coordinates for each vertex are numbered from zero, in the same order as they were specified when the object was created. The parameter *index* specifies which one to use.

ERRORS

DoTextureUVIndex will fail if *index* is less than zero.

[WARNING - value out of range]

DEFAULTS

The default value for *DoTextureUVIndex* is 0.

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureUVWIndex(3D)

NAME

DoTextureUVWIndex – Create a texture attribute object that specifies which **uvw** coordinates of a primitive object to use

SYNOPSIS

C:

```
DtObject DoTextureUVWIndex(index)
DtInt index;
```

Fortran:

```
INTEGER*4 DOTWI(INDEX)
INTEGER*4 INDEX
```

DESCRIPTION

DoTextureUVWIndex creates a texture attribute object that specifies which set of **uvw** coordinates in a primitive object to use when doing 3-dimensional texture mapping. Each primitive object can have more than one **uvw** coordinate per vertex. The **uvw** coordinates for each vertex are numbered from zero, in the same order as they were specified when the object was created. The parameter *index* specifies which one to use.

ERRORS

DoTextureUVWIndex will fail if *index* is less than zero.

[WARNING - value out of range]

DEFAULTS

The default value for *DoTextureUVWIndex* is 0.

SEE ALSO

DoTextureMapDiffuseColor(3D), DoTextureMapEnviron(3D), DoTextureMapBump(3D), DoTextureMapTranspIntens(3D), DoTextureUVIndex(3D)

NAME

DoTorus – Create a torus primitive object

SYNOPSIS

C:

```
DtObject DoTorus(bigradius, smallradius)
DtReal bigradius;
DtReal smallradius;
```

Fortran:

```
INTEGER*4 DOTOR(BIGRAD, SMLRAD)
REAL*8 BIGRAD
REAL*8 SMLRAD
```

DESCRIPTION

DoTorus creates a torus (donut-shaped) primitive object. The parameter *bigradius* specifies the distance from the center (within the hole) of the torus to the center of the ring. The parameter *smallradius* specifies the radius of the cross-section of the ring itself. The torus is centered at the origin and the perimeter of the large circle is in the XZ-plane.

SEE ALSO

DoSubDivSpec(3D)

NAME

DoTransformMatrix – Create a transformation matrix geometric transformation object

SYNOPSIS

C:

```
DtObject DoTransformMatrix(matrix, comptype)
DtMatrix4x4 matrix;
DtCompType comptype;
```

Fortran:

```
INTEGER*4 DOTMX(MATRIX, COMTYP)
REAL*8 MATRIX(4,4)
INTEGER*4 COMTYP
```

DESCRIPTION

DoTransformMatrix creates a transformation matrix geometric transformation object. *DoTransformMatrix* is one of the geometric transformation objects whose value is used to determine the current transformation matrix attribute. Unlike other geometric transformation objects that always preconcatenate their values with the current transformation matrix attribute so they apply first, *DoTransformMatrix* allows the user to decide how it affects the current transformation matrix attribute.

The parameter *matrix* specifies an arbitrary modeling transformation to be combined with the current transformation matrix attribute. The parameter *comptype* specifies the way *matrix* will be combined with the current transformation matrix. Possible values for *comptype* are:

DcReplace <DCREPL>
: *matrix* replaces the current transformation matrix.

DcPreConcatenate <DCPREC>
: *matrix* is preconcatenated to the current transformation matrix; i.e., this transformation applies before any previous transformations.

DcPostConcatenate <DCPSTC>
: *matrix* is postconcatenated to the current transformation matrix; i.e., this transformation applies after any previous transformations.

Points are represented as column vectors when doing matrix multiplications. Thus postconcatenation means premultiplication and preconcatenation means postmultiplication.

The parameter *matrix* can be an arbitrary 4x4 matrix, but singular matrices (non-invertible) will cause fatal errors.

SEE ALSO

DoLookAtFrom(3D), DoRotate(3D), DoScale(3D), DoShear(3D), DoTranslate(3D)

NAME

DoTranslate – Create a translation geometric transformation object

SYNOPSIS

C:

```
DtObject DoTranslate(x, y, z)
DtReal x, y, z;
```

Fortran:

```
INTEGER*4 DOXLT(X, Y, Z)
REAL*8 X, Y, Z
```

DESCRIPTION

DoTranslate creates a translation geometric transformation object that modifies the current transformation matrix attribute. The translation can be in the X-, Y-, or Z-directions relative to the current origin. The parameters *x*, *y*, and *z* are three floating point values for the X-, Y-, and Z-displacements.

SEE ALSO

DoLookAtFrom(3D), DoRotate(3D), DoScale(3D), DoShear(3D), DoTransformMatrix(3D)

NAME

DoTranspColor – Create a transparent color primitive attribute object

SYNOPSIS

C:

```
DtObject DoTranspColor(colormodel, color)
DtColorModel colormodel;
DtReal color[ ];
```

Fortran:

```
INTEGER*4 DOTC(COLMOD, COLOR)
INTEGER*4 COLMOD
REAL*8 COLOR(*)
```

DESCRIPTION

DoTranspColor creates a transparent color primitive attribute object. The parameter *colormodel* specifies the color model. The parameter *color* specifies the color of the light that can be transmitted through the surface. In effect, the transparent color of an object can be thought of as its filter color.

The transparent color is one of three aspects of the transparent component of a surface. The other components are the transparent color's intensity and the applicability of transparent components to a surface.

The use of this attribute during rendering is renderer-dependent. Some renderers which support transparancy do not support a transparent color.

DEFAULTS

The default *DoTranspColor* is (*DcRGB*, (1.0, 1.0, 1.0)).

SEE ALSO

DoTranspIntens(3D), DoTranspOrientColor(3D),DoTranspOrientExp(3D), DoTranspOrientIntens(3D),DoTranspOrientSwitch(3D),DoTranspSwitch(3D)

NAME

DoTranspIntens – Create a transparent intensity primitive attribute object

SYNOPSIS

C:

```
DtObject DoTranspIntens(intensity)
DtReal intensity;
```

Fortran:

```
INTEGER*4 DOTI(INTENS)
REAL*8 INTENS
```

DESCRIPTION

DoTranspIntens creates a transparent intensity primitive attribute object. The parameter *intensity* specifies the intensity of the response of a surface to light transmitted through it. The parameter *intensity* is usually a value between 0.0 and 1.0 that signifies how much of the transmitted light actually gets through the surface.

The transparent intensity is one of three aspects of the transparent component of a surface. The other aspects are the transparent color and the applicability of transparent components to a surface.

DEFAULTS

The default *DoTranspIntens* is 0.5.

SEE ALSO

DoTranspColor(3D), DoTranspOrientColor(3D), DoTranspOrientExp(3D), DoTranspOrientIntens(3D), DoTranspOrientSwitch(3D), DoTranspSwitch(3D)

NAME

DoTranspOrientColor – Create a transparent color primitive attribute object for orientation dependent transparency

SYNOPSIS

C:

```
DtObject DoTranspOrientColor(colormodel, color)
DtColorModel colormodel;
DtReal color[];
```

Fortran:

```
INTEGER*4 DOTOC(COLMOD, COLOR)
INTEGER*4 COLMOD
REAL*8 COLOR(*)
```

DESCRIPTION

DoTranspOrientColor creates a transparent color primitive attribute object for orientation dependent transparency.

Orientation dependent transparency means that the transparency of an object varies depending on the orientation of the object and the view direction. The transparency can also vary across the object if it is not completely flat.

The parameter *color* specifies the transparent color to be used for subsequent objects when the object is oriented such that the object normal is perpendicular to the viewing direction. For other object orientations the object transparent color for orientation dependent transparency is calculated as follows:

transpcolor = orientcolor + (color - orientcolor)((I dot N) ** orientexp)*

where *orientcolor* is the value specified by *DoTranspOrientColor*, *color* is the value specified by *DoTranspColor*, *orientexp* is the value specified by *DoTranspOrientExp*, *N* is the object normal, and *I* is the view direction (or the incoming ray direction for ray-tracing).

DEFAULTS

The default *DoTranspOrientColor* is (*DcRGB*, 1.0,1.0,1.0.).

SEE ALSO

DoTranspOrientSwitch(3D), DoTranspOrientIntens(3D), DoTranspOrientExp(3D), DoTranspIntens(3D), DoTranspColor(3D)

NAME

DoTranspOrientExp – Create a transparent exponent primitive attribute object for orientation dependent transparency

SYNOPSIS

C:

```
DtObject DoTranspOrientExp(exponent)
DtReal exponent;
```

Fortran:

```
INTEGER*4 DOTOE(EXP)
REAL*8 EXP
```

DESCRIPTION

DoTranspOrientExp creates a transparent exponent primitive attribute object for orientation dependent transparency.

Orientation dependent transparency means that the transparency of an object varies depending on the orientation of the object and the view direction. The transparency can also vary across the object if it is not completely flat.

The parameter *exponent* specifies the transparent exponent to be used for subsequent objects when the object is oriented such that the object normal is not perpendicular to the viewing direction. See *DoTranspOrientColor* and *DoTranspOrientIntens* for details of the formulas that use the transparent exponent.

DEFAULTS

The default *DoTranspOrientExp* is 1.0.

SEE ALSO

DoTranspOrientSwitch(3D), DoTranspOrientColor(3D), DoTranspOrientIntens(3D)

NAME

DoTranspOrientIntens – Create a transparent intensity primitive attribute object for orientation dependent transparency

SYNOPSIS

C:

```
DtObject DoTranspOrientIntens(intensity)
DtReal intensity;
```

Fortran:

```
INTEGER*4 DOTOI(INTENS)
REAL*8 INTENS
```

DESCRIPTION

DoTranspOrientIntens creates a transparent intensity primitive attribute object for orientation dependent transparency.

Orientation dependent transparency means that the transparency of an object varies depending on the orientation of the object and the view direction. The transparency can also vary across the object if it is not completely flat.

The parameter *intensity* specifies the transparent intensity to be used for subsequent objects when the object is oriented such that the object normal is perpendicular to the viewing direction. For other object orientations the object transparent intensity for orientation dependent transparency is calculated as follows:

transpintens = orientintens + (intens - orientintens)((I dot N) ** orientexp)*

where *orientintens* is the value specified by *DoTranspOrientIntens*, *intens* is the value specified by *DoTranspIntens <DOTI>*, *orientexp* is the value specified by *DoTranspOrientExp <DOTOE>*, *N* is the object normal, and *I* is the viewing direction (or the incoming ray direction for raytracing).

DEFAULTS

The default *DoTranspOrientIntens* is 0.5.

SEE ALSO

DoTranspOrientSwitch(3D), DoTranspOrientColor(3D), DoTranspOrientExp(3D), DoTranspIntens(3D), DoTranspColor(3D)

NAME

DoTranspOrientSwitch – Create a primitive attribute object for enabling/disabling orienation dependent transparency

SYNOPSIS

C:

```
DtObject DoTranspOrientSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOTOS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoTranspOrientSwitch creates a primitive attribute object for enabling/disabling orientation dependent transparency.

Orientation dependent transparency means that the transparency of an object varies depending on the orientation of the object and the view direction. The transparency can also vary across the object if it is not completely flat.

The attributes *DoTranspIntens* <*DOTI*> and *DoTranspColor* <*DOTC*> are used when the viewing direction coincides with the object normal. The values of the attributes *DoTranspOrientIntens* <*DOTOI*> and *DoTranspOrientColor* <*DOTOC*> are used when the object normal is perpendicular to the viewing direction. Otherwise a combination of both sets is used. See *DoTranspOrientIntens* and *DoTranspOrientColor* for more details.

The parameter *switchvalue* specifies whether subsequent primitive objects have an orientation dependent transparent component.

DcOn <*DCON*>
: The object will render itself without orientation dependent transparency, regardless of the settings for other orientation dependent transparency attributes.

DcOff <*DCOFF*>
: The object will render itself with orientation dependent transparency whenever possible.

DEFAULTS

The default *DoTranspOrientSwitch* is *DcOff* <*DCOFF*>.

SEE ALSO

DoTranspOrientIntens(3D), DoTranspOrientColor(3D), DoTranspOrientExp(3D), DoTranspIntens(3D), DoTranspColor(3D)

NAME

DoTranspSwitch – Create a transparent switch primitive attribute object

SYNOPSIS

C:

```
DtObject DoTranspSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
INTEGER*4 DOTS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DoTranspSwitch creates a transparent switch primitive attribute object. The parameter *switchvalue* specifies whether the surfaces of subsequent primitive objects have a transparent component. Possible values are:

DcOff <DCOFF>
The object will render itself without transparency regardless of the settings for transparent color and transparent intensity.

DcOn <DCON>
The object will render itself with transparency whenever possible.

The transparent switch is one of three aspects of the transparent component of a surface. The other components are the transparent color and the transparent color's intensity.

DEFAULTS

The default for *DoTranspSwitch* is *DcOff <DCOFF>*.

SEE ALSO

DoTranspColor(3D), DoTranspIntens(3D), DoTranspOrientColor(3D), DoTranspOrientExp(3D), DoTranspOrientIntens(3D), DoTranspOrientSwitch(3D)

NAME

DoTriangleList – Create a triangle list primitive object

SYNOPSIS

C:

```
DtObject DoTriangleList(colormodel, vertextype, trianglecount, vertices)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt trianglecount;
DtReal vertices[];
```

Fortran:

```
INTEGER*4 DOTRIL(COLMOD, VTXTYP, TRICNT, VTXS)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 TRICNT
REAL*8 VTXS(*)
```

DESCRIPTION

DoTriangleList creates a primitive object that defines a list of independent triangles. The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *trianglecount* contains the number of triangles in the list. The parameter *vertices* is an array of vertex data. It must contain data for 3**trianglecount* vertices.

The geometric normal to each triangle is calculated using the right-hand rule (when the fingers of the right hand point in the order of the vertices, the thumb points in the direction of the normal). If no vertex normals are provided by the application program, or facet shading is specified, Doré computes geometric normals to find backfacing triangles for backface culling and incident and reflected light angles for shading.

SEE ALSO

DoTriangleMesh(3D), VertexTypes(3D)

NAME

DoTriangleMesh – Create a triangle mesh primitive object

SYNOPSIS

C:

```
DtObject DoTriangleMesh(colormodel, vertextype, vertexcount, vertices,
                        trianglecount, triangles, smoothflag)
DtColorModel colormodel;
DtVertexType vertextype;
DtInt vertexcount;
DtReal vertices[];
DtInt trianglecount;
DtInt triangles[];
DtFlag smoothflag;
```

Fortran:

```
INTEGER*4 DOTRIM(COLMOD, VTXTYP, VTXCNT, VTXS,
                 TRICNT, TRIS, SMOOFL)
INTEGER*4 COLMOD
INTEGER*4 VTXTYP
INTEGER*4 VTXCNT
REAL*8 VTXS(*)
INTEGER*4 TRICNT
INTEGER*4 TRIS(*)
INTEGER*4 SMOOFL
```

DESCRIPTION

DoTriangleMesh creates a triangle mesh primitive object that defines a collection of triangles, normally interconnected.

Mesh objects allow the user to specify a collection of vertices in space and connect the vertices into a collection of other geometric entities, usually forming a connected surface. Triangle meshes are generally used to approximate a smooth surface.

The parameter *colormodel* specifies the color model used if the vertices contain color information for shading purposes. The parameter *vertextype* specifies the exact nature of the vertices. See the appendix on vertex types for more information on this parameter. The parameter *vertexcount* specifies the total number of vertices in the mesh. The parameter *vertices* is an array of vertex data. The parameter *trianglecount* specifies the total number of triangles in the mesh. The parameter *triangles* is a one-dimensional array containing *trianglecount* triplets of integers. Each triplet is an ordered list of mesh vertex numbers for the three coordinates of the triangle.

If vertex normals are not provided, the parameter *smoothflag* specifies that the vertex normal for shading purposes will be the average of the geometric normals from a vertex's adjacent surfaces. This assumes that the triangle mesh defines a smooth surface.

The geometric normal to each triangle is calculated using the right-hand rule (when the fingers of the right hand point in the order of the vertices, the thumb points in the direction of the normal). If no vertex normals are provided, or facet shading is specified, Doré computes geometric normals to find back-facing triangles for back-face culling and incident and reflected light angles for shading.

ERRORS

DoTriangleMesh must be called with at least three vertices.

[WARNING - invalid parameter]

DoTriangleMesh will issue an warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

DoTriangleMesh will fail if an invalid vertex list pointer is specified.

[WARNING - value out of range]

SEE ALSO

DoPolygon(3D), DoPolygonMesh(3D), DoSimplePolygon(3D), DoSimplePolygonMesh(3D), DoTriangleList(3D), VertexTypes(3D)

NAME

DoVarLineList – Create a variable line list primitive object

SYNOPSIS

C:

```
DtObject DoVarLineList(colormodel, linecount, vertlocs, vertnorms, vertcolors)
DtColorModel colormodel;
DtInt linecount;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
```

Fortran:

```
INTEGER*4 DOVLNL(COLMOD, LINCNT, VTXLOC, VTXNRM, VTXCLR)
INTEGER*4 COLMOD
INTEGER*4 LINCNT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
```

DESCRIPTION

DoVarLineList creates a primitive object that defines a variable list of independent lines. The main difference between a variable line list and a line list is that the variable line list does not copy the vertex data into its own data space. It simply maintains pointers to the vertex data (locations, normals and colors) in user space. A call to *DpUpdVarLineList* <*DPUVLL*> informs a variable line list object that the user data has changed.

The parameter *colormodel* specifies the color model to be used when defining vertex colors (if any). The parameter *linecount* contains the number of lines in the list. The parameter *vertlocs* is an array of three dimensional vertices containing the locations of 2**linecount* vertices. The parameter *vertnorms* is an array of vertex normals. If vertex normals are not available use *DcNullPtr* <*DCNULL*>. The parameter *vertcolors* is an array of vertex colors. If vertex colors are not available use *DcNullPtr* <*DCNULL*>.

If no vertex normals are provided, backface culling has no effect for lines because they have no surface on which to define a geometric normal.

SEE ALSO

DpUpdVarLineList(3D), DoLineList(3D), DoVarPointList(3D), DoVarTriangleMesh(3D), DoVarSimplePolygonMesh(3D)

NAME

DoVarPointList – Create a variable point list primitive object

SYNOPSIS

C:

```
DtObject DoVarPointList(colormodel, pointcount, vertlocs, vertnorms, vertcolors)
DtColorModel colormodel;
DtInt pointcount;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
```

Fortran:

```
INTEGER*4 DOVPTL(COLMOD, PNTCNT, VTXLOC, VTXNRM, VTXCLR)
INTEGER*4 COLMOD
INTEGER*4 PNTCNT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
```

DESCRIPTION

DoVarPointList creates a primitive object that defines a variable list of independent points. The main difference between a variable point list and a point list is that the variable point list does not copy the vertex data into its own data space. It simply maintains pointers to the vertex data (locations, normals and colors) in user space. A call to *DpUpdVarPointList* <*DPUVPL*> informs a variable point list object that the user data has changed.

The parameter *colormodel* specifies the color model to be used when defining vertex colors (if any). The parameter *pointcount* contains the number of points in the list. The parameter *vertlocs* is an array of three dimensional vertices containing the locations of *pointcount* vertices. The parameter *vertnorms* is an array of vertex normals. If vertex normals are not available use *DcNullPtr* <*DCNULL*>. The parameter *vertcolors* is an array of vertex colors. If vertex colors are not available use *DcNullPtr* <*DCNULL*>.

If vertex normals are not provided, backface culling has no effect for points because they have no surface on which to define a geometric normal.

SEE ALSO

DpUpdVarPointList(3D), DoPointList(3D), DoVarLineList(3D), DoVarTriangleMesh(3D), DoVarSimplePolygonMesh(3D)

NAME

DoVarSimplePolygonMesh – Create a variable simple polygon mesh primitive object

SYNOPSIS

C:

```
DtObject DoVarSimplePolygonMesh(colormodel, vertexcount, vertlocs,
                vertnorms, vertcolors, polygoncount, contours, vertexlist,
                shape, smoothflag)
DtColorModel colormodel;
DtInt vertexcount;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
DtInt polygoncount;
DtInt contours[];
DtInt vertexlist[];
DtShapeType shape;
DtFlag smoothflag;
```

Fortran:

```
INTEGER*4 DOVSPM(COLMOD, VTXCNT, VTXLOC, VTXNRM, VTXCLR,
                PLYCNT, CONTURS, VTXLST, SHAPE, SMOOFL)
INTEGER*4 COLMOD
INTEGER*4 VTXCNT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
INTEGER*4 PLYCNT
INTEGER*4 CONTURS(PLYCNT)
INTEGER*4 VTXLST(*)
INTEGER*4 SHAPE
INTEGER*4 SMOOFL
```

DESCRIPTION

DoVarSimplePolygonMesh creates a primitive object that defines a variable collection of simple polygons, normally interconnected. The main difference between a variable simple polygon mesh and a simple polygon mesh is that the variable mesh does not copy the vertex data into its own data space. It simply maintains pointers to the vertex data (locations, normals and colors) in user space. A call to *DpUpdVarSimplePolygonMesh* <*DPUVSM*> informs an object that the user data has changed. Each simple polygon must be as described in *DoSimplePolygon*, with the additional restriction that the shape must not be *DcComplex* <*DCCPLX*>.

The parameter *colormodel* specifies the color model used if vertex colors are specified. The parameter *vertexcount* specifies the number of vertices. The parameter *vertlocs* is an array of vertex locations. The parameter *vertnorms* is an array of vertex normals. If vertex normals are not available use *DcNullPtr* <*DCNULL*>. The parameter *vertcolors* is an array of vertex colors. If vertex colors are not available use *DcNullPtr* <*DCNULL*>.

The parameter *polygoncount* specifies the number of polygons in the mesh. Simple polygons are each composed of one contour. The parameter *contours* is an array of *polygoncount* integers, each of which specifies the number of vertices in the associated polygon.

The parameter *vertexlist* is an array of integers that index the *vertices* array. The first polygon in the mesh begins with the vertex specified by *vertexlist[0]*, and continues using the vertices specified by the following entries in *vertexlist*. The last vertex of a contour is implicitly connected to its first vertex. The next polygon begins with the vertex specified next in the *vertexlist* array.

The parameter *shape* hints at the geometry of the polygons in the mesh. For these polygons, correctly specifying the shape of the polygon will result in optimal polygon decomposition when necessary. Incorrectly specifying the shape may result in an incorrect or aborted decomposition. The possible values for *shape* are:

DcConvex <DCCNVX>
Path is wholly convex.

DcConcave <DCCNCV>
Path may be concave but not self-intersecting.

Note that the shape must *not* be *DcComplex <DCCPLX>*.

The parameter *smoothflag* specifies that if vertex normals are not provided, the average normal of the geometric normals from a vertex's adjacent surfaces constitute the normal for vertex shading. (This assumes the simple polygon mesh defines a smooth surface.)

The geometric normal to each triangle is calculated using the right-hand rule (when the fingers of the right hand point in the order of the vertices, the thumb points in the direction of the normal). If vertex normals are not available, or facet shading is specified, Doré computes geometric normals to calculate back-facing polygons for backface culling and incident and reflected light angles for shading.

ERRORS

DoVarSimplePolygonMesh must be called with the a valid specification for *shape.*

[WARNING - invalid parameter]

DoVarSimplePolygonMesh must be called with at least three vertices.

[WARNING - insufficient information]

DoVarSimplePolygonMesh will issue a warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

DoVarSimplePolygonMesh will fail if the polygon is specified as *DcComplex.*

[WARNING - value out of range]

SEE ALSO

DpUpdVarSimplePolygonMesh(3D), DoSimplePolygonMesh(3D), DoVarLineList(3D), DoVarPointList(3D), DoVarTriangleMesh(3D)

NAME

DoVarTriangleMesh – Create a variable triangle mesh primitive object

SYNOPSIS

C:

```
DtObject DoVarTriangleMesh(colormodel, vertexcount, vertlocs, vertnorms,
                           vertcolors, trianglecount, triangles, smoothflag)
DtColorModel colormodel;
DtInt vertexcount;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
DtInt trianglecount;
DtInt triangles[];
DtFlag smoothflag;
```

Fortran:

```
INTEGER*4 DOVTRM(COLMOD, VTXCNT, VTXLOC, VTXNRM, VTXCLR,
                 TRICNT, TRIS, SMOOFL)
INTEGER*4 COLMOD
INTEGER*4 VTXCNT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
INTEGER*4 TRICNT
INTEGER*4 TRIS(3*TRICNT)
INTEGER*4 SMOOFL
```

DESCRIPTION

DoVarTriangleMesh creates a primitive object that defines a variable collection of triangles, normally interconnected.

Mesh objects allow the user to specify a collection of vertices in space and connect the vertices into a collection of other geometric entities, usually forming a connected surface. Variable triangle meshes are generally used to approximate a smooth surface.

The main difference between a variable triangle mesh and a triangle mesh is that a variable triangle mesh does not make a copy of the vertex data into its own data space. It simply maintains pointers to the vertex data (locations, normals and colors) in user space. A call to *DpUpdVarTriangleMesh* <*DPUVTM*> informs a variable triangle mesh object that the user data has changed.

The parameter *colormodel* specifies the color model used if vertex colors are provided. The parameter *vertexcount* specifies the total number of vertices in the mesh. The parameter *vertlocs* is an array of vertex locations. The parameter *vertnorms* is an array of vertex normals. If vertex normals are not available use *DcNullPtr* <*DCNULL*>. The parameter *vertcolors* is an array of vertex colors. If vertex colors are not available use *DcNullPtr* <*DCNULL*>. The parameter *trianglecount* specifies the total number of triangles in the mesh. The parameter *triangles* is a one-dimensional array containing *trianglecount* triplets of integers. Each triplet is an ordered list of mesh vertex numbers for the three coordinates of the triangle.

If vertex normals are not provided, the parameter *smoothflag* specifies that the vertex normal for shading purposes will be the average of the geometric normals from a vertex's adjacent surfaces. This assumes that the triangle mesh defines a smooth surface.

The geometric normal to each triangle is calculated using the right-hand rule (when the fingers of the right hand point in the order of the vertices, the thumb points in the direction of the normal). If vertex normals are not available, or facet shading is specified, Doré computes geometric normals to calculate back-facing triangles for backface culling and incident and reflected light angles for shading.

ERRORS

DoVarTriangleMesh must be called with at least three vertices.

[WARNING - invalid parameter]

DoVarTriangleMesh will issue a warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

SEE ALSO

DpUpdVarTriangleMesh(3D), DoTriangleMesh(3D), DoVarLineList(3D), DoVarPointList(3D), DoVarSimplePolygonMesh(3D)

NAME

DoView – Create a view organizational object

SYNOPSIS

C:

```
DtObject DoView()
```

Fortran:

```
INTEGER*4 DOVW()
```

DESCRIPTION

DoView creates a new view. A view is an organizational object that describes a scene.

A view contains a display group and a definition group. The display group contains all of the objects to be displayed when the view is updated. The definition group contains the available cameras that view the scene and the available lights that illuminate it. In addition, views also contain attributes that affect aspects of the final images. These aspects are:

The active camera, one of the cameras in the definition group

The background color

The clear flag, whether to clear to background before rendering

The render style

The view boundary, in frame coordinates

SEE ALSO

DvInqDefinitionGroup(3D), DvInqDisplayGroup(3D), DvSetActiveCamera(3D), DdSetBackgroundColor(3D), DvSetClearFlag(3D), DvSetRendStyle(3D), DvSetBoundary(3D), DvUpdate(3D)

NAME

DpUpdVarLineList – Update a variable line list primitive object

SYNOPSIS

C:

```
DtObject DpUpdVarLineList(object, linecount, vertlocs, vertnorms, vertcolors)
DtObject object;
DtInt linecount;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
```

Fortran:

```
INTEGER*4 DPUVLL(OBJECT, LINCNT, VTXLOC, VTXNRM, VTXCLR)
INTEGER*4 OBJECT
INTEGER*4 LINCNT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
```

DESCRIPTION

DpUpdVarLineList updates a variable line list primitive object.

The parameter *object* is the handle to the object.

The parameter *linecount* specifies the number of lines in the vertex data arrays. A value of *0* indicates that the number of lines has not changed.

The parameter *vertlocs* is an array of vertex locations. If the vertex locations have not been modified use *DcNullPtr <DCNULL>*.

The parameter *vertnorms* is an array of vertex normals. If the vertex normals have not been modified use *DcNullPtr <DCNULL>*. *vertnorms* must be a null pointer if vertex normals were not provided when the object was created with *DoVarLineList <DOVLNL>*.

The parameter *vertcolors* is an array of vertex colors. If the vertex colors have not been modified use *DcNullPtr <DCNULL>*. *vertcolors* must be a null pointer if vertex colors were not provided when the object was created with *DoVarLineList <DOVLNL>*.

ERRORS

DpUpdateVarLineList will fail if an invalid object is specified.

[WARNING - invalid or deleted object]

DpUpdateVarLineList will fail if vertex normals are specified for an object that did not possess them originally.

[WARNING - value out of range]

DpUpdateVarLineList will fail if vertex colors are specified for an object that did not possess them originally.

[WARNING - value out of range]

SEE ALSO

DoVarLineList(3D)

NAME

DpUpdVarPointList – Update a variable point list primitive object

SYNOPSIS

C:

```
DtObject DpUpdVarPointList(object, linecount, vertlocs, vertnorms,
                           vertcolors)
DtObject object;
DtInt linecount;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
```

Fortran:

```
INTEGER*4 DPUVPL(OBJECT, LINCNT, VTXLOC, VTXNRM, VTXCLR)
INTEGER*4 OBJECT
INTEGER*4 LINCNT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
```

DESCRIPTION

DpUpdVarPointList updates a variable point list primitive object.

The parameter *object* is the handle to the object.

The parameter *linecount* specifies the number of lines in the vertex data arrays. A value of *0* indicates that the number of lines has not changed.

The parameter *vertlocs* is an array of vertex locations. If the vertex locations have not been modified use *DcNullPtr <DCNULL>*.

The parameter *vertnorms* is an array of vertex normals. If the vertex normals have not been modified use *DcNullPtr <DCNULL>*. *vertnorms* must be a null pointer if vertex normals were not provided when the object was created with *DoVarPointList <DOVPTL>*.

The parameter *vertcolors* is an array of vertex colors. If the vertex colors have not been modified use *DcNullPtr <DCNULL>*. *vertcolors* must be a null pointer if vertex colors were not provided when the object was created with *DoVarPointList <DOVPTL>*.

ERRORS

DpUpdateVarPointList will fail if an invalid object is specified.

[WARNING - invalid or deleted object]

DpUpdateVarPointList will fail if vertex normals are specified for an object that did not possess them originally.

[WARNING - value out of range]

DpUpdateVarPointList will fail if vertex colors are specified for an object that did not possess them originally.

[WARNING - value out of range]

SEE ALSO

DoVarPointList(3D)

NAME

DpUpdVarSimplePolygonMesh – Update a variable simple polygon mesh primitive object

SYNOPSIS

C:

```
DtObject DpUpdVarSimplePolygonMesh(object, vertlocs, vertnorms,
                                   vertcolors, shape, decompose, recompute_norms)
DtObject object;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
DtShapeType shape;
DtFlag decompose;
DtFlag recompute_norms;
```

Fortran:

```
INTEGER*4 DPUVSM(OBJECT, VTXLOC, VTXNRM, VTXCLR, SHAPE,
                 DECOMP, CMPNRM)
INTEGER*4 OBJECT
REAL*8 VTXLOC(*)
REAL*8 VTXNRM(*)
REAL*8 VTXCLR(*)
INTEGER*4 SHAPE
INTEGER*4 DECOMP
INTEGER*4 CMPNRM
```

DESCRIPTION

DpUpdVarSimplePolygonMesh updates a variable simple polygon mesh primitive object.

The parameter *object* is the handle to the object.

The parameter *vertlocs* is an array of vertex locations. If the vertex locations have not been modified use *DcNullPtr <DCNULL>*. The number of vertices, the number of polygons and the mesh connectivity cannot be modified.

The parameter *vertnorms* is an array of vertex normals. If the vertex normals have not been modified use *DcNullPtr <DCNULL>*. *vertnorms* must be a null pointer if vertex normals were not provided when the object was created with *DoVarSimplePolygonMesh <DOVSPM>*.

The parameter *vertcolors* is an array of vertex colors. If the vertex colors have not been modified use *DcNullPtr <DCNULL>*. *vertcolors* must be a null pointer if vertex colors were not provided when the object was created with *DoVarSimplePolygonMesh <DOVSPM>*.

The parameter *shape* hints at the geometry of the polygons in the mesh. The possible values for *shape* are:

DcConvex <DCCNVX>
: Path is wholly convex.

DcConcave <DCCNCV>
: Path may be concave but not self-intersecting.

Note that the shape must *not* be *DcComplex <DCCPLX>*. If the vertex locations have been modified then the shape may not be the same as when the object was created. The parameter *decompose* specifies whether the polygons need to be redecomposed for rendering if the vertex locations have been modified.

The parameter *recompute_norms* specifies whether normals should be recomputed if *vertlocs* is not *DcNull* <*DCNULL*>. The vertex normals will only be recomputed if *smoothflag* was set when the object was created with *DoVarSimplePolygonMesh* <*DOVSPM*>, and if the shading needs them.

ERRORS

DpUpdateVarSimplePolygonMesh will fail if the polygon shape is specified as *DcComplex*.

[WARNING - value out of range]

DpUpdateVarSimplePolygonMesh will issue an warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

DpUpdateVarSimplePolygonMesh will fail if an invalid object is specified.

[WARNING - invalid or deleted object]

DpUpdateVarSimplePolygonMesh will fail if vertex normals are specified for an object that did not possess them originally.

[WARNING - value out of range]

DpUpdateVarSimplePolygonMesh will fail if vertex colors are specified for an object that did not possess them originally.

[WARNING - value out of range]

SEE ALSO

DoVarSimplePolygonMesh(3D)

NAME

DpUpdVarTriangleMesh – Update a variable triangle mesh primitive object

SYNOPSIS

C:

```
DtObject DpUpdVarTriangleMesh(object, vertlocs, vertnorms, vertcolors,
                              recompute_norms)
DtObject object;
DtReal vertlocs[];
DtReal vertnorms[];
DtReal vertcolors[];
DtFlag recompute_norms;
```

Fortran:

```
INTEGER*4 DPUVTM(OBJECT, VTXLOC, VTXNRM, VTXCLR, CMPNRM)
INTEGER*4 OBJECT
REAL*8 VTXLOC(3,*)
REAL*8 VTXNRM(3,*)
REAL*8 VTXCLR(*,*)
INTEGER*4 CMPNRM
```

DESCRIPTION

DpUpdVarTriangleMesh updates a variable triangle mesh primitive object.

The parameter *object* is the handle to the object.

The parameter *vertlocs* is an array of vertex locations. If the vertex locations have not been modified use *DcNullPtr <DCNULL>*. The number of vertices, the number of triangles and the mesh connectivity cannot be modified.

The parameter *vertnorms* is an array of vertex normals. If the vertex normals have not been modified use *DcNullPtr <DCNULL>*. *vertnorms* must be a null pointer if vertex normals were not provided when the object was created with *DoVarTriangleMesh <DOVTRM>*.

The parameter *vertcolors* is an array of vertex colors. If the vertex colors have not been modified use *DcNullPtr <DCNULL>*. *vertcolors* must be a null pointer if vertex colors were not provided when the object was created with *DoVarTriangleMesh, <DOVTRM>*.

The parameter *recompute_norms* specifies whether normals should be recomputed if *vertlocs* is not null. The vertex normals will only be recomputed if *smoothflag* was set when the object was created with *DoVarTriangleMesh <DOVTRM>*, and if the shading needs them.

ERRORS

DpUpdateVarTriangleMesh will issue a warning if a zero vertex normal is calculated by Doré.

[WARNING - triangle normals sum to 0 at vertex, check for back to back triangles]

DpUpdateVarTriangleMesh will fail if an invalid object is specified.

[WARNING - invalid or deleted object]

DpUpdateVarTriangleMesh will fail if vertex normals are specified for an object that did not possess them originally.

[WARNING - value out of range]

DpUpdateVarTriangleMesh will fail if vertex colors are specified for an object that did not possess them originally.

[WARNING - value out of range]

SEE ALSO

DoVarTriangleMesh(3D)

NAME

DsCompBoundingVol – Compute dimensions of a bounding volume for an object

SYNOPSIS

C:

```
DtVolume *DsCompBoundingVol(volume, object)
DtVolume *volume;
DtObject object;
```

Fortran:

```
INTEGER*4 DSCBV(VOLUME, OBJECT)
INTEGER*4 OBJECT
REAL*8 VOLUME(6)
```

DESCRIPTION

DsCompBoundingVol computes a bounding volume for an object, usually a primitive object or a group. The function computes a right rectangular volume that tightly encloses the given object and is parallel to the object's modeling coordinate system. The computed volume is returned in the parameter *volume*. The parameter *object* specifies the object whose bounding volume is to be computed. The function returns the same pointer (now the pointer to the computed volume) that was passed with the function call.

ERRORS

DsCompBoundingVol will fail if the object handle is invalid.

[WARNING - invalid or deleted object]

DsCompBoundingVol will fail if Doré is already performing a traversal of the database.

[WARNING - traversal already in progress]

SEE ALSO

DoBoundingVol(3D), DoBoundingVolSwitch(3D), DoMinBoundingVolExt(3D)

NAME

DsExecuteObj – Execute an object immediately

SYNOPSIS

C:

```
void DsExecuteObj(object)
DtObject object;
```

Fortran:

```
CALL DSEO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsExecuteObj causes the immediate execution of an object. This functionality provides true conditional execution by creating callback objects that affect group traversal by choosing which objects (sub-trees) to execute.

Objects executed by callbacks are executed with the same method with which the callback was executed.

ERRORS

DsExecuteObj will fail if passed an invalid object.

[WARNING - invalid or deleted object]

DsExecuteObj will fail if no traversal of the database is in progress.

[WARNING - no traversal in progress]

SEE ALSO

DoCallback(3D)

NAME

DsExecutionAbort – Abort traversal of the current method

SYNOPSIS

C:

```
void DsExecutionAbort()
```

Fortran:

```
CALL DSEA()
```

DESCRIPTION

DsExecutionAbort aborts traversal of the current method. Callback objects may conditionally affect traversal of display or definition hierarchies into which they are placed by calling *DsExecutionAbort*. This causes the entire traversal to halt.

The traversal will halt only upon return from the callback function that called *DsExecutionAbort*. In other words, the callback function will be allowed to finish, but no other objects will be executed after it in the same traversal.

ERRORS

DsExecutionAbort will fail if called outside of a traversal method; the call to *DsExecutionAbort* will be ignored.

[WARNING - no traversal in progress]

SEE ALSO

DoCallback(3D), DsExecuteObj(3D), DsExecutionReturn(3D)

NAME

DsExecutionReturn – Abort traversal of the current group

SYNOPSIS

C:

```
void DsExecutionReturn()
```

Fortran:

```
CALL DSER()
```

DESCRIPTION

DsExecutionReturn causes traversal of the current group to abort. Callback objects may conditionally affect traversal of display or definition hierarchies into which they are placed by calling *DsExecutionReturn*. This halts traversal of the current group in which they are found.

The execution of this group will halt only upon return from the callback function that called *DsExecutionReturn*. In other words, the callback function will be allowed to finish, but no other objects in the same group will be executed after it.

ERRORS

DsExecutionReturn will fail if called outside of a traversal method; the call to *DsExecutionReturn* will be ignored.

[WARNING - no traversal in progress]

SEE ALSO

DoCallback(3D), DsExecuteObj(3D), DsExecutionAbort(3D)

NAME

DsFileRasterRead – Read raster information from a file

SYNOPSIS

C:

```
DtInt DsFileRasterRead(filename, width, height, depth, type, data)
DtPtr filename;
DtInt *width;
DtInt *height;
DtInt *depth;
DtRasterType *type;
DtPtr *data;
```

Fortran:

```
INTEGER*4 DSFRSR(FILENM, FLEN, WIDTH, HEIGHT, DEPTH,
        TYPE, DATA)
INTEGER*4 FLEN
CHARACTER*FLEN FILENM
INTEGER*4 WIDTH
INTEGER*4 HEIGHT
INTEGER*4 DEPTH
INTEGER*4 TYPE
INTEGER*4 DATA
```

DESCRIPTION

DsFileRasterRead reads raster information from a file. This infomation can be used to create a raster object with *DoRaster <DORS>*.

The parameter *filename* is the name of the file containing the raster data.

The file can contain information for a one-, two- or three-dimensional raster. The return parameters *width*, *height*, and *depth* are the dimensions of the raster.

The return parameter *type* indicates the type and format of the information being extracted. Possible values for *type* are:

DcRasterRGB <DCRRGB>
Each point in the raster has red, green and blue information.

DcRasterRGBA <DCRRA>
Each point in the raster has red, green, blue and alpha information.

DcRasterRGBAZ <DCRRAZ>
Each point in the raster has red, green, blue, alpha and z information.

DcRasterRGBZ <DCRRZ>
Each point in the raster has red, green, blue and z information.

DcRasterA <DCRA>
Each point in the raster has alpha information.

DcRasterZ <DCRZ>
Each point in the raster has z information.

The return parameter *data* is a pointer to the extracted raster data. *DsFileRasterRead* will allocate the space for the data, but it is up to the application to deallocate the space when the data is no longer needed, for example through the delete callback mechanism of *DoRaster <DORS>*.

DsFileRasterRead returns -1 if an error occurs (see below), otherwise *DsFileRasterRead* returns 0.

FORTRAN SPECIFIC

The parameter *FILENM* is a filename *FLEN* characters long.

ERRORS

DsFileRasterRead will fail if the file is not a standard Doré raster file.

[WARNING - value out of range. 'rastertype' required as first attribute]

DsFileRasterRead will fail if the file cannot be opened.

[WARNING - io error. Could not open file *filename*]

SEE ALSO

DoRaster(3D), DoFileRaster(3D)

NAME

DsHoldObj – Place a hold on an object

SYNOPSIS

C:

```
DtObject DsHoldObj(object)
DtObject object;
```

Fortran:

```
INTEGER*4 DSHO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsHoldObj places a hold on an object specified by the parameter *object*, and it returns the object handle.

There are two ways objects keep track of how they are being used. First, every object maintains a *reference count* that indicates how many times it is referenced by other objects. Second, the object maintains a hold flag that indicates whether the user has any references to it. The user has explicit control over this flag. If an object's reference count is 0, and its hold flag is *DcFalse* <*DCFALS*>, then it is considered deletable: the object is removed from the database, its storage area is destroyed, and its object handle is no longer valid.

When an object is created, its reference count is 0, and its hold flag is *DcFalse* <*DCFALS*>. If the user adds the object to a group, its reference count is automatically incremented, and it is *not* considered deletable by the database. If the user wishes to reference an object's handle outside of the current database and it is not referenced within the database, the user should call the *DsHoldObj* function with the object handle. This causes the object's hold flag to be set to true. The corresponding routine *DsReleaseObj* <*DSRO*> releases the hold on the object when the user has no further need of it. The user can determine whether a hold has been placed on the object by calling *DsInqHoldObj* <*DSQHO*>.

A common way to call the *DsHoldObj* function is:

```
object = DsHoldObj(DoXXXX());
```

where the *DoXXXX()* call creates and returns an object of type XXXX. A hold is placed on the object through the *DsHoldObj* call. One can also create an object through a separate call to a *DoXXXX* routine and then call the *DsHoldObj* routine with the object directly:

```
object = DoXXXX();
DsHoldObj(object);
```

The user may also call this function in the following manner:

```
DsHoldObj(object=DoXXXX());
```

This will result in creating the object, setting the *object* variable to its handle, and placing a hold on the object.

ERRORS

DsHoldObj will fail if passed an invalid object.

[WARNING - invalid or deleted object]

SEE ALSO

DsInqHoldObj(3D), DsReleaseObj(3D)

NAME

DsInitializeSystem – Initialize Doré

SYNOPSIS

C:

```
void DsInitializeSystem(processors)
DtInt processors;
```

Fortran:

```
CALL DSINIT(PROS)
INTEGER*4 PROS
```

DESCRIPTION

DsInitializeSystem initializes the Doré system. This must be the first Doré call made by an application. Successive calls to this function are ignored unless preceded by a call to *DsTerminateSystem <DSTERM>*.

The *processors* parameter specifies the multiprocessor usage of this run of Doré. The allowable values are:

0	use only one processor
positive number	run in multiprocessor mode, with the specified number of processors

Note that a value of 1 tells Doré to run in a multiprocessor mode, but with only one processor. This mode is useful for debugging purposes, but is generally less efficient than specifying 0 (true uniprocessor operation).

SEE ALSO

DsTerminateSystem(3D)

NAME

DsInputValue – Post a value to an input slot

SYNOPSIS

C:

```
void DsInputValue(slot, value)
DtObject slot;
DtReal value;
```

Fortran:

```
CALL DSIV(SLOT, VALUE)
INTEGER*4 SLOT
REAL*8 VALUE
```

DESCRIPTION

DsInputValue posts a floating point value to an input slot organizational object. An input slot is an entry point for a *DtReal* value and provides the primary interface for connecting input events and values to Doré.

Each input slot has a group of 0 or more valuator callback objects. A valuator is a callback object that depends upon input slots. *DsInputValue* is used primarily to achieve dynamic displays by using these input values to affect object attributes or their geometry. The input values can be generated by application programs or by asynchronous external events such as dial or mouse movement events.

When an input value arrives at an input slot, all valuator callback objects dependent upon it are called.

The parameter *slot* specifies the input slot to which the *DtReal* in parameter *value* is to be posted.

ERRORS

DsInputValue will fail is passed an invalid slot handle.

[WARNING - invalid slot handle]

SEE ALSO

DoCallBack(3D), DoInputSlot(3D), DsInqValuatorGroup(3D), DsUpdateAllViews(3D), DsValuatorSwitch(3D)

NAME

DsInqClassId – Return the class identifier of a named class

SYNOPSIS

C:

```
DtInt DsInqClassId (class_name)
DtPtr class_name;
```

Fortran:

```
INTEGER*4 DSQCI(CLSNME, N)
CHARACTER*N CLSNME
```

DESCRIPTION

DsInqClassId returns the class identifier of the class with the string name *class_name*. If *class_name* is not the name of a class installed in Doré then *DsInqClassId* returns the value -1.

By convention, a class is given the same name as the C Language name of the routine used to create an object of that class. For example, an object created with *DoPolygon* is of class "DoPolygon".

DsInqClassId is often used to find the class identifier of a user-defined class.

FORTRAN SPECIFIC

The *CLSNME* parameter is the class name string, which is *N* characters long.

NAME

DsInqCurrentMethod – Return the current method being executed

SYNOPSIS

C:

DtInt DsInqCurrentMethod ()

Fortran:

INTEGER*4 DSQCM()

DESCRIPTION

DsInqCurrentMethod returns the current method as a number. The methods are named in *dore.h* <*DOREMETHODS*>. If there is not a current method then *DsInqCurrentMethod* returns *DcMethodNull* <*DCNULL*>.

SEE ALSO

DoCallback(3D)

NAME

DsInqErrorMessage – Return an error message

SYNOPSIS

C:

```
DtReadStatus DsInqErrorMessage(errornumber, bufbytes, buf, severity)
DtInt errornumber, bufbytes;
char buf[ ];
DtErrorStatus *severity;
```

Fortran:

```
INTEGER*4 DSQEM(ERRNUM, BUFSIZ, BUF, SEVERT)
INTEGER*4 ERRNUM, BUFSIZ
CHARACTER*BUFSIZ BUF
INTEGER*4 SEVERT
```

DESCRIPTION

DsInqErrorMessage copies an error message into user-supplied space. The parameter *errornumber* specifies the error message. The parameter *bufbytes* specifies the size of the user-supplied space. The parameter *buf* specifies the address of that space.

The parameter *severity* is a returned error status specifying the severity associated with the error. The severity describes the type of error and is one of the following values:

DcErrorMinor <DCEMNR>
Program will procede with only minor effects likely.

DcErrorSevere <DCESEV>
Program can procede but results are unpredictable.

DcErrorFatal <DCEFAT>
Program cannot procede. If the default error handler is being used, it will exit the program. If a user-specified error handler is being used, it should try to exit gracefully.

DsInqErrorMessage is intended for use by user-supplied error handling routines installed with *DsSetErrorVars <DSSEV>*. Possible return values are:

DcReadOk <DCROK>
The error message was succesfully read and written into the user's space.

DcReadTrunc <DCRTRN>
The error message was too large to fit in the given space and was truncated.

DcReadUnsuc <DCRFAL>
The given error was not found.

SEE ALSO

DsSetErrorVars(3D)

NAME

DsInqErrorVars – Return current error file and error handler

SYNOPSIS

C:

```
void DsInqErrorVars(errorfile, errorhandler)
DtInt *errorfile;
DtPFI *errorhandler;
```

Fortran:

```
CALL DSQEV(ERRFIL, ERRHND)
INTEGER*4 ERRFIL
INTEGER*4 ERRHND
```

DESCRIPTION

DsInqErrorVars returns the current error file descriptor and error handler. The parameter *errorfile* specifies the location of the current error file descriptor. The parameter *errorhandler* specifies the location of the current error handler. See *DsSetErrorVars* for a more detailed description of the Doré error handling system.

SEE ALSO

DsInqErrorMessage(3D), DsSetErrorVars(3D)

NAME

DsInqExeDepthLimit – Return the maximum depth to which objects will be executed

SYNOPSIS

C:

```
DtInt DsInqExeDepthLimit()
```

Fortran:

```
INTEGER*4 DSQEDL()
```

DESCRIPTION

DsInqExeDepthLimit returns the maximum allowed object execution depth, i.e., the number of descending generations a parent object may have.

SEE ALSO

DsSetExeDepthLimit(3D)

NAME

DsInqHoldObj – Return whether a hold has been placed on the object

SYNOPSIS

C:

```
DtFlag DsInqHoldObj(object)
DtObject object;
```

Fortran:

```
INTEGER*4 DSQHO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsInqHoldObj returns whether a hold has been placed on the object specified by the parameter *object*. See *DsHoldObj* for details on hold objects.

ERRORS

DsInqHoldObj will fail if passed an invalid object; the returned value is undefined.

[WARNING - invalid or deleted object]

SEE ALSO

DsHoldObj(3D), DsReleaseObj(3D)

NAME

DsInqMethodId – Return the method identifier of a named method

SYNOPSIS

C:

```
DtInt DsInqMethodId(method_name)
DtPtr method_name;
```

Fortran:

```
INTEGER*4 DSQMI(METNME, N)
CHARACTER*N METNME
```

DESCRIPTION

DsInqMethodId returns the integer identifier of the method with the string name *method_name*. If *method_name* is not the name of a method installed in Doré, *DsInqMethodId* returns the value -1.

FORTRAN SPECIFIC

The *METNME* parameter is the method name string, which is *N* characters long.

SEE ALSO

DsInqCurrentMethod(3D)

NAME

DsInqNumRenderers – Return the number of installed renderers

SYNOPSIS

C:

```
DtInt DsInqNumRenderers()
```

Fortran:

```
INTEGER *4 DSQNR()
```

DESCRIPTION

DsInqNumRenderers returns the number of available renderers in the current Doré System. *DsInqNumRenderers* can be used in conjunction with *DsInqRendererNames* <*DSQRNS*> to obtain a list of available renderers.

SEE ALSO

DsInqRendererNames(3D)

NAME

DsInqObj – Return an object specified by name

SYNOPSIS

C:

```
DtObject DsInqObj(object_name_type, object_name, object_type)
DtNameType object_name_type;
DtPtr object_name;
DtInt object_type;
```

Fortran:

```
INTEGER*4 DSQOI(OBJNUM, OBJTYP)
INTEGER*4 OBJNUM
INTEGER*4 OBJTYP

INTEGER*4 DSQOS(OBJSTR, N, OBJTYP)
INTEGER*4 N
CHARACTER*N OBJSTR
INTEGER*4 OBJTYP
```

DESCRIPTION

DsInqObj returns a handle to an object. The parameter *object_name* specifies the name of the object. The parameter *object_type* contains an integer indicating the type of the object being queried. The integer representing a particular object type can be determined using *DsInqClassId(class_name)* where *class_name* is the C name of the *DoXXXXX* function used to create an object. For example, *DsInqClassId("DoScale")* returns the integer object type value for a scale object. Possible values for *object_name_type* are:

DcNameInteger
An integer value name.

DcNameString
A string value name.

FORTRAN SPECIFIC

Function *DSQOI* queries the handle of an object when the name is of type *DCNINT*. *OBJNUM* is the integer name of the object and *OBJTYP* is an integer value indicating the type of object being queried. The integer representing a particular object type can be determined using *DSQCI(CLSNME,N)* where *CLSNME* is the C name of the *DoXXXXX* function used to create an object and *N* is the length of the C name. For example, *DSQCI('DoScale', 7)* returns the integer object type value for a scale object.

Function *DSQOI* queries the handle of an object when the object name is of the type *DCNSTR*. *OBJSTR* is a string of *N* characters containing the name of the object. *OBJTYP* is an integer value indicating the type of object.

ERRORS

DsInqObj will fail if the *name_type* argument is invalid.

[WARNING - invalid parameter]

SEE ALSO

DsInqClassId(3D), DsInqObjName(3D), DsInqObjType(3D)

NAME

DsInqObjName – Return an object's name from its object handle

SYNOPSIS

C:

```
void DsInqObjName(object, object_name_type, object_name)
DtObject object;
DtNameType *object_name_type;
DtPtr object_name;
```

Fortran:

```
INTEGER*4 DSQONT(OBJECT)
INTEGER*4 OBJECT

INTEGER*4 DSQONI(OBJECT)
INTEGER*4 OBJECT

CALL DSQONS(OBJECT, FNAME, LENGTH)
INTEGER*4 OBJECT
INTEGER*4 LENGTH
CHARACTER*LENGTH FNAME
```

DESCRIPTION

DsInqObjName returns an object's name given its object handle. The parameter *object* specifies the object. The parameter *object_name* holds a pointer to the object name. The parameter *object_name_type* holds the type of the object name. Possible return values for *object_name_type* are:

DcNameInteger
An integer value name.

DcNameString
A string value name. Object names are delimited by a trailing null.

DcNameNone
No name given.

Doré maintains a system wide object name table subdivided by object type. The *DsInqObjName* function can obtain the name of any object in this table.

FORTRAN SPECIFIC

DSQONT returns the object type (class) of the object *OBJECT*.

DSQONI returns the name of the object *OBJECT*. This function should be used when the object name is of type *DCNINT* (it's an integer).

DSQONS returns the name of the object *OBJECT*. This function should be used when the object name is of type *DCNSTR* (it's a string). The string name is returned in *FNAME* and is *LENGTH* characters long.

ERRORS

DsInqObjName will fail if the object handle is invalid.

[WARNING - invalid or deleted object]

SEE ALSO

DsInqObj(3D)

NAME

DsInqObjStatus – Check the existence of an object

SYNOPSIS

C:

```
DtObjectStatus DsInqObjStatus(object)
DtObject object;
```

Fortran:

```
INTEGER*4 DSQVOS(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsInqObjStatus returns the status of the object specified by the parameter *object*. Possible return values are:

DcObjectValid <DCOVLD>
The object is exists.

DcObjectDeleted <DCODEL>
The object has been deleted.

DcObjectInvalid <DCOINV>
The object does not exist.

SEE ALSO

DsInqObj(3D), DsInqObjName(3D)

NAME

DsInqObjType – Return an object's type from its object handle

SYNOPSIS

C:

```
DtInt DsInqObjType(object)
DtObject object;
```

Fortran:

```
INTEGER*4 DSQOT(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsInqObjType finds an object's type given its object handle. The parameter *object* specifies the object.

When an object is created, it is assigned a type. This type is identical to the class identifier of the class of which the object is an instance. The class identifier for a particular class can be queried with *DsInqClassId <DSQCI>*.

ERRORS

DsInqObjType will fail if the object handle is invalid; a -1 is returned.

[WARNING - invalid or deleted object]

SEE ALSO

DsInqClassId(3D)

NAME

DsInqRendererId – Return the renderer identifier of a named renderer

SYNOPSIS

C:

```
DtInt DsinqRendererid(renderer_name)
DtPtr renderer_name;
```

Fortran:

```
INTEGER*4 DSQRI(RENNME, N)
CHARACTER*N RENNME
```

DESCRIPTION

DsInqRendererId returns the integer identifier of the renderer with the string name *renderer_name*. If *renderer_name* is not the name of a renderer installed in the current Doré system, *DsInqRendererId* returns -1.

FORTRAN SPECIFIC

The *RENNME* parameter is the renderer name string, which is *N* characters long.

SEE ALSO

DsInqNumRenderers(3D), DsInqRendererNames(3D), DvInqRendStyle(3D)

NAME

DsInqRendererNames – Return a list of the names of the installed renderers

SYNOPSIS

C:

```
void DsInqRendererNames(names)
DtPtr names[];
```

Fortran:

```
DSQRNS(NAMES, LENGTH)
CHARACTER NAMES(*)*LENGTH
INTEGER*4 LENGTH
```

DESCRIPTION

DsInqRendererNames returns in the array *names* the names of the available renderers in the current Doré System. The length of the array *names* is assumed to be long enough to contain the complete list. *DsInqNumRenderers* <*DSQNR*> returns the number of renderers and should be used to determine the necessary size of the array *names*. For example:

```
DtPtr *names;
DtInt cnt;

cnt = DsInqNumRenderers();
names = (DtPtr *)malloc(cnt*sizeof(DtPtr));
DsInqRendererNames(names);
```

FORTRAN SPECIFIC

The parameter *LENGTH* specifies the length of each of the character arrays passed to *DSQRNS*. Renderer names that are longer than *LENGTH* will be truncated to fit in the specified size.

SEE ALSO

DsInqNumRenderers(3D)

NAME

DsInqSafeFlag – Query the safe flag

SYNOPSIS

C:

DtFlag DsInqSafeFlag()

Fortran:

INTEGER*4 DSQSF()

DESCRIPTION

DsInqSafeFlag queries the safe flag. The safe flag is a debugging tool for the application programmer. When the safe flag is set to *DcTrue* <*DCTRUE*>, all deleted objects are treated so that any private data associated with the object is released as usual but the type field in the object header is set to *DcObjectDeleted* <*DCODEL*>. Whenever such an object is accessed, an error is generated and the user can determine why the illegal reference was made.

If the flag is set to *DcFalse* <*DCFALS*>, all deleted objects are freed immediately, both the object header and any private data. If the safe flag is changed from *DcTrue* <*DCTRUE*> to *DcFalse* <*DCFALS*>, all object headers of type *DcObjectDeleted* <*DCODEL*> are freed.

An application should not refer to an object's handle once the application has released the object using *DsReleaseObj* <*DSRO*>.

SEE ALSO

DsSetSafeFlag(3D)

NAME

DsInqValuatorGroup – Return the handle for an input slot's valuator group

SYNOPSIS

C:

```
DtObject DsInqValuatorGroup(slot)
DtObject slot;
```

Fortran:

```
INTEGER*4 DSQVG(SLOT)
INTEGER*4 SLOT
```

DESCRIPTION

DsInqValuatorGroup returns the handle for an input slot's valuator group. A valuator group is part of every input slot. The valuator group contains the object handles of all the valuator callback objects associated with the input slot. When a value arrives at an input slot, all callback objects in its valuator group are triggered in order. Users are free to edit that group, with the group editing functions, to arrange their valuator callbacks in the order that they want them triggered. When an input slot is triggered, all non-callback objects found in its valuator group are ignored and the callback functions that it refers to are called. Those functions must have the following C format:

```
void UserValuatorFunction(data, slot, value)
        DtPtr data;      /* if data is a pointer */
        Dt32Bits data;   /* if data is a value */
        DtObject slot;
        DtReal  value;
```

And the following Fortran format:

```
SUBROUTINE VALUATORFCN(DATA, SLOT, VALUE)
        INTEGER*4 DATA
        INTEGER*4 SLOT
        REAL*8 VALUE
```

The parameter *data* is the same data with which the valuator function was installed, the parameter *slot* is the slot at which the new value arrived, and the parameter *value* is the actual input value.

Valuators are usually used to achieve dynamic displays. There are two major ways to achieve dynamic displays. The most straightforward method is for the user valuator function to use the group editing routines to make lasting changes to the database. If the desired result is to redefine a primitive, the valuator function only needs to create the new primitive and overwrite the old one using *DgReplaceObj <DGRO>* or *DgReplaceObjInGroup <DRGOG>*. The same technique can be used to modify an attribute. This can also be accomplished by creating an inline group that contains only the element to be affected. This inline group would be instanced at appropriate places in the database allowing several subtrees to inherit that attribute.

Another way to achieve dynamics is for the user valuator functions to modify some private data that will also be made known to user-supplied callback functions installed as group elements. When executed, these callback group elements can then perform *DsExecuteObj <DSEO>* calls on "DoXXXX" functions with arguments that depend on the private data being maintained by user valuator functions. Hence, immediate execution of those functions is achieved without storing them in the data base.

After an input value has arrived at an input slot and all valuators dependent upon it have been triggered, the views containing the affected objects need to be updated in order to efficiently achieve dynamics.

A C example of creating a valuator is :

```
DgAddObjToGroup(DsInqValuatorGroup(slot), callbackobj);
```

ERRORS

DsInqValuatorGroup will fail if the input slot handle is invalid.

[WARNING - invalid slot handle]

SEE ALSO

DgReplaceObj(3D), DgReplaceObjInGroup(3D), DoCallback(3D), DoInLineGroup(3D), DoInputSlot(3D), DsInputValue(3D), DsUpdateAllViews(3D), DsValuatorSwitch(3D)

NAME

DsInqVersion – Inquire the string describing the current version

SYNOPSIS

C:

```
DsInqVersion (version)
DtPtr *version;
```

Fortran:

```
INTEGER*4 DSQVER(VERSN, LENGTH)
INTEGER*4 LENGTH
CHARACTER*LENGTH VERSN
```

DESCRIPTION

DsInqVersion returns a printable string describing the current version of Doré in the parameter *version*.

FORTRAN SPECIFIC

The return string is specified by *VERSN* and is *LENGTH* bytes long.

NAME

DsPrintObj – Print information about an object

SYNOPSIS

C:

```
void DsPrintObj(object)
DtObject object;
```

Fortran:

```
CALL DSPO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsPrintObj prints information about the object specified by parameter *object*. The object may be any object, including groups. Each object has its own method for printing information about itself.

ERRORS

DsPrintObj will fail if the object is deleted or non-existent.

[WARNING - invalid or deleted object]

NAME

DsRasterUpdate – Update a raster object

SYNOPSIS

C:

```
DsRasterUpdate(raster)
DtObject raster;
```

Fortran:

```
DSRSU(RASTER)
INTEGER*4 RASTER
```

DESCRIPTION

DsRasterUpdate notifies Doré that the data for a raster object has changed.

The parameter *raster* is a handle to the raster object.

ERRORS

DsRasterUpdate will fail if *raster* is not a valid raster object.

[WARNING - invalid or deleted object]

SEE ALSO

DoRaster(3D)

NAME

DsRasterWrite – Write a raster object to a file

SYNOPSIS

C:

```
DtInt DsRasterWrite(raster, filename)
DtObject raster;
DtPtr filename;
```

Fortran:

```
INTEGER*4 DSRSW(RASTER, FNAME, FLEN)
INTEGER*4 RASTER
INTEGER*4 FLEN
CHARACTER*FLEN FNAME
```

DESCRIPTION

DsRasterWrite takes a raster object and writes it out to a file.

The parameter *raster* is a handle to the raster object. The parameter *filename* is the name of the file.

DsRasterWrite returns -1 if an error occurs (see below), otherwise *DsRasterWrite* returns 0.

FORTRAN SPECIFIC

The parameter *FNAME* is an filename that is *FLEN* characters long.

ERRORS

DsRasterWrite will fail if *raster* is not a valid raster object.

[WARNING - invalid or deleted object]

DsRasterWrite will fail if the raster object data is not one of the standard Doré types.

[WARNING - unimplemented function. Cannot write DcRasterSpecial.]

DsRasterWrite will fail if the file cannot be opened.

[WARNING - io error. Could not open file *filename* for writing]

SEE ALSO

DoRaster(3D)

NAME

DsReleaseObj – Release a hold previously placed on an object

SYNOPSIS

C:

```
void DsReleaseObj(object)
DtObject object;
```

Fortran:

```
CALL DSRO(OBJECT)
INTEGER*4 OBJECT
```

DESCRIPTION

DsReleaseObj releases a previously placed hold on an object and deletes the object if it is not referenced by any other objects. See *DsHoldObj*.

ERRORS

DsReleaseObj will fail if passed an invalid object.

[WARNING - invalid or deleted object]

SEE ALSO

DsHoldObj(3D), DsInqHoldObj(3D)

NAME

DsSetErrorVars – Specify an error file and error handler

SYNOPSIS

C:

```
void DsSetErrorVars(errorfile, errorhandler)
DtInt errorfile;
DtPFI errorhandler;
```

Fortran:

```
CALL DSSEV(ERRFIL, ERRHND)
INTEGER*4 ERRFIL
EXTERNAL ERRHND
```

DESCRIPTION

DsSetErrorVars installs an application error file descriptor and error handler. The parameter *errorfile* specifies the file descriptor to which any error messages should be sent. The value 2 is usually *standard error*, i.e., the terminal, and is also the default error file.

The parameter *errorhandler* specifies the error handling procedure to be called when errors are detected. If this parameter is *DcNullPtr <DCNULL>*, the default error handler prints the error information to *errorfile* and, if the error is fatal, it exits the program.

A user supplied error handling procedure in C should have the following format:

```
MyErrorFunction(errorfile, errornumber, funcname, errorstring)
DtInt errorfile;
DtPtr funcname;
DtInt errornumber;
DtPtr errorstring;
```

FORTRAN SPECIFIC

A user supplied error handling procedure in Fortran should have the following format:

```
ERRFUN(ERRFIL, ERRNUM, FCNNAM, FCNNML, ERRSTR, ERRSTL)
INTEGER*4 ERRFIL
INTEGER*4 ERRNUM
INTEGER*4 FCNNML
INTEGER*4 ERRSTL
CHARACTER*FCNNML FCNNAM
CHARACTER*ERRSTL ERRSTR
```

Where *FCNNAM* is a function name *FCNNML* characters long and *ERRSTR* is an error string *ERRSTL* characters long.

SEE ALSO

DsInqErrorMessage(3D)

NAME

DsSetExeDepthLimit – Specify the maximum allowed depth to which objects will be executed

SYNOPSIS

C:

```
void DsSetExeDepthLimit(limit)
DtInt limit;
```

Fortran:

```
CALL DSSEDL(LIMIT)
INTEGER*4 LIMIT
```

DESCRIPTION

DsSetExeDepthLimit is used to set the maximum allowed object execution depth, i.e., the number of descending generations a parent object may have. The parameter *limit* is the depth in generations.

ERRORS

DsSetExeDepthLimit will fail if the execution limit is less than or equal to 0.

[WARNING - value out of range]

DEFAULTS

The default execution depth limit is 20.

SEE ALSO

DsInqExeDepthLimit(3D)

NAME

DsSetObjName – Set the name of an object

SYNOPSIS

C:

```
void DsSetObjName(object, name_type, object_name, replace)
DtObject object;
DtNameType name_type;
DtPtr object_name;
DtFlag replace;
```

Fortran:

```
CALL DSSOND(OBJECT)
INTEGER*4 OBJECT

CALL DSSONI(OJBECT, OBJNUM, REPL)
INTEGER*4 OBJECT
INTEGER*4 OBJNUM
INTEGER*4 REPL

CALL DSSONS(OBJECT, OBJSTR, N, REPL)
INTEGER*4 OBJECT
INTEGER*4 REPL
INTEGER*4 N
CHARACTER*N OBJSTR
```

DESCRIPTION

DsSetObjName sets the name of an object specified by the parameter *object*. There is a system wide object name table. This table is subdivided by object type. *DsSetObjName* assigns a name of *object_name* to any object. The parameter *name_type* specifies the type of *object_name*. Possible values for *name_type* are:

DcNameInteger
An integer value name.

DcNameString
A string value name.

DcNameNone
No name given; the object name will be ignored. The user may pass an *object_name* of *DcNullPtr* here.

No two objects of the same object type may have the same object name. The *replace* flag signals whether to report an error if an object of the same type already has this name (*DcOff* <*DCOFF*>) or to set the name in all cases and remove the name from any object (of this type) that has the proposed name (*DcOn* <*DCON*>).

FORTRAN SPECIFIC

DSSOND sets the name of *OBJECT* to *DCNNON*. No name is given; the object name will be ignored.

DSSONI sets the name of *OBJECT* to the integer *OBJNUM*. This function gives an object an integer value name.

DSSONS sets the name of *OBJECT* to the string of *N* characters in *OBJSTR*. This function gives an object a string value name.

ERRORS

DsSetObjName will fail if the object is invalid.

[WARNING - invalid or deleted object]

DsSetObjName will fail if the name is already in use and the replace flag is not *DcOn <DCON>*.

[WARNING - object name already in use]

SEE ALSO

DsInqObj(3D), DsInqObjName(3D)

NAME

DsSetSafeFlag – Set the safe flag

SYNOPSIS

C:

```
void DsSetSafeFlag(flag)
DtFlag flag ;
```

Fortran:

```
CALL DSSSF(FLAG)
INTEGER*4 FLAG
```

DESCRIPTION

DsSetSafeFlag sets the safe flag to the value specified by the parameter *flag*. See *DsInqSafeFlag* for further explanation.

DEFAULTS

The default value of the safe flag is *DcTrue* <*DCTRUE*>.

SEE ALSO

DsInqSafeFlag(3D)

NAME

DsTerminateSystem – Terminate Doré

SYNOPSIS

C:

```
void DsTerminateSystem()
```

Fortran:

```
CALL DSTERM()
```

DESCRIPTION

DsTerminateSystem terminates the Doré system and is the normal way of exiting Doré. The graphical command stream will be flushed. This command is not reversible; no state is saved.

ERRORS

DsTerminateSystem always works.

SEE ALSO

DsInitializeSystem(3D)

NAME

DsTextureUVCount – Set the number of **uv** sets in the vertex type specification

SYNOPSIS

C:

```
DtInt DsTextureUVCount(count)
DtInt count;
```

Fortran:

```
INTEGER*4 DSTUVC(COUNT)
INTEGER*4 COUNT
```

DESCRIPTION

DsTextureUVCount is a system function that returns a value to be ORed with a base vertex type to create a new vertex type that includes **uv** coordinates. The new vertex type is specified when primitive objects requiring the specification of vertices are created. The parameter *count* specifies the number of **uv** coordinates that are present for each vertex in the vertex list. For example, if the vertices of a primitive were to contain locations, normals and 2 **uv** coordinates, the vertex type would be set with: *DcLocNrm | DsTextureUVCount(2)*. Each vertex in this example requires ten DtReal values. Thus, for vertex n the array would look like:

```
vertices[10*n+0] = x
vertices[10*n+1] = y
vertices[10*n+2] = z
vertices[10*n+3] = nx
vertices[10*n+4] = ny
vertices[10*n+5] = nz
vertices[10*n+6] = u1
vertices[10*n+7] = v1
vertices[10*n+8] = u2
vertices[10*n+9] = v2
```

where the position of vertex n is (x,y,z), the normal is (nx, ny, nz), the first **uv** coordinate is (u1,v1) and the second **uv** coordinate is (u2,v2).

Note that a vertex type may include **uvw** coordinates in addition to **uv** coordinates. In that case both *DsTextureUVWCount <DSTWC>* and *DsTextureUVCount* are used in conjunction with the base vertex type, and in the vertex array all the **uv** coordinates of a vertex come before the **uvw** coordinates for that vertex.

SEE ALSO

DsTextureUVWCount(3D), VertexTypes(3D)

NAME

DsTextureUVWCount – Set the number of **uvw** coordinates in the vertex type specification

SYNOPSIS

C:

```
DtInt DsTextureUVWCount(count)
DtInt count;
```

Fortran:

```
INTEGER*4 DSTWC(COUNT)
INTEGER*4 COUNT
```

DESCRIPTION

DsTextureUVWCount is a system function that returns a value to be ORed with a base vertex type to create a new vertex type that includes **uvw** coordinates. The new vertex type is specified when primitive objects requiring the specification of vertices are created. The parameter *count* specifies the number of **uvw** coordinates that are present for each vertex in the vertex list. For example, if the vertices of a primitive were to contain locations, normals and two **uvw** coordinates, the vertex type would be set with: *DcLocNrm | DsTextureUVWCount(2)*. Each vertex in this example requires twelve DtReal values. Thus, for vertex n the array would look like:

```
vertices[10*n+0] = x
vertices[10*n+1] = y
vertices[10*n+2] = z
vertices[10*n+3] = nx
vertices[10*n+4] = ny
vertices[10*n+5] = nz
vertices[10*n+6] = u1
vertices[10*n+7] = v1
vertices[10*n+8] = w1
vertices[10*n+9] = u2
vertices[10*n+10] = v2
vertices[10*n+11] = w2
```

where the position of vertex n is (x,y,z), the normal is (nx, ny, nz), the first **uvw** coordinate is (u1,v1,w1) and the second **uvw** coordinate is (u2,v2,w2).

Note that a vertex type may include **uv** coordinates in addition to **uvw** coordinates. In that case both *DsTextureUVCount <DSTUVC>* and *DsTextureUVWCount* are used in conjunction with the base vertex type, and in the vertex array all the **uv** coordinates of a vertex come before the **uvw** coordinates for that vertex.

SEE ALSO

DsTextureUVCount(3D), VertexTypes(3D)

NAME

DsUpdateAllViews – Update all views

SYNOPSIS

C:

```
void DsUpdateAllViews()
```

Fortran:

```
CALL DSUAV()
```

DESCRIPTION

DsUpdateAllViews causes all views to update themselves.

SEE ALSO

DdUpdate(3D), DvUpdate(3D)

NAME

DsValuatorSwitch – Enable or disable valuators

SYNOPSIS

C:

```
void DsValuatorSwitch(switchvalue)
DtSwitch switchvalue;
```

Fortran:

```
CALL DSVS(SWVAL)
INTEGER*4 SWVAL
```

DESCRIPTION

DsValuatorSwitch sets the system valuator switch. The parameter *switchvalue* specifies whether valuator callbacks are currently allowed to execute.

A valuator is a callback object that depends on input slots. Input slots are entry points for *DtReal* type values. These input slots are the primary interface for connecting input events and values to Doré.

When the value of the valuator switch is *DcOff* <*DCOFF*>, all inputs through input slots are blocked. When the value of the valuator switch is *DcOn* <*DCON*>, all blocked inputs are flushed. This mechanism allows applications to edit the database without worrying about simultaneous editing by valuators responding to external events.

DEFAULTS

The default *DsValuatorSwitch* is *DcOn* <*DCON*>.

SEE ALSO

DoCallback(3D), DoInputSlot(3D), DsInputValue(3D), DsInqValuatorGroup(3D)

NAME

DvInqActiveCamera – Return the active camera for a view

SYNOPSIS

C:

```
DtObject DvInqActiveCamera(view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQAC(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqActiveCamera queries the active camera to be used with a view specified by the parameter *view* when rendering. The active camera specifies which camera's attributes will be used to render a view.

ERRORS

DvInqActiveCamera will fail if the view handle is null; the returned value is *DcNullObject <DCNULL>*.

[WARNING - invalid view handle]

SEE ALSO

DvSetActiveCamera(3D)

NAME

DvInqBackgroundColor – Return the background color of a view

SYNOPSIS

C:

```
void DvInqBackgroundColor(view, colormodel, color)
DtObject view;
DtColorModel *colormodel;
DtReal color[];
```

Fortran:

```
CALL DVQBC(VIEW, COLMOD, COLOR)
INTEGER*4 VIEW
INTEGER*4 COLMOD
REAL*8 COLOR(*)
```

DESCRIPTION

DvInqBackgroundColor queries the background color of the view specified by the parameter *view*. The parameter *colormodel* specifies the color model used. The parameter *color* specifies the color to which a view is cleared when the view's clear flag is set and the view is updated.

ERRORS

DvInqBackgroundColor will fail if the view handle is invalid.

[WARNING - invalid view handle]

SEE ALSO

DvSetBackgroundColor(3D)

NAME

DvInqBoundary – Return the view boundary

SYNOPSIS

C:

```
void DvInqBoundary(view, boundary)
DtObject view;
DtVolume *boundary;
```

Fortran:

```
CALL DVQB(VIEW, BNDRY)
INTEGER*4 VIEW
REAL*8 BNDRY(6)
```

DESCRIPTION

DvInqBoundary returns the view boundary for the view specified by the parameter *view*. The view boundary parameter *boundary* specifies the position and amount of space occupied by the view within a frame.

ERRORS

DvInqBoundary will fail if the view handle is invalid.

[WARNING - invalid view handle]

SEE ALSO

DvSetBoundary(3D)

NAME

DvInqClearFlag – Return the clear flag of a view

SYNOPSIS

C:

```
DtFlag DvInqClearFlag(view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQCF(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqClearFlag returns the value of *clearflag* for the view specified by the parameter *view*. If *DcTrue* <*DCTRUE*> is returned, the view is cleared to its background color every time the view is updated. If *DcFalse* <*DCFALS*> is returned, the view is not cleared.

ERRORS

DvInqClearFlag will fail if the view handle is invalid.

[WARNING - invalid view handle]

SEE ALSO

DvSetClearFlag(3D)

NAME

DvInqDefinitionGroup – Return the definition group for a view

SYNOPSIS

C:

```
DtObject DvInqDefinitionGroup(view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQDG(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqDefinitionGroup returns the handle for a definition group of the view specified by the parameter *view*. The definition group is the part of every view that contains the objects that define the studio objects. When a view is updated, all objects in its definition group are executed sequentially. Users may edit the definition group to arrange objects in the desired order of execution. The objects in a view's definition group may be any of the studio objects, studio attribute objects, or other groups of the same. Objects that are not studio object types are ignored during definition traversal.

ERRORS

DvInqDefinitionGroup will fail if the view handle is invalid; the returned value is *DcNullObject* <*DCNULL*>.

[WARNING - invalid view handle]

SEE ALSO

DfInqViewGroup(3D), DoView(3D), DvInqDisplayGroup(3D), DvSetActiveCamera(3D), DvUpdate(3D)

NAME

DvInqDisplayGroup – Return the handle for a view's display group

SYNOPSIS

C:

```
DtObject DvInqDisplayGroup(view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQIG(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqDisplayGroup returns the handle for a display group of the view specified by the parameter *view*. The display group is the part of every view that contains the objects that are to be displayed when the view is updated. When a view is updated, all objects in its display group are executed in order. Users may edit the display group to arrange objects in the desired order of execution.

ERRORS

DvInqDisplayGroup will fail if the view handle is invalid; the returned value is *DcNullObject <DCNULL>*.

[WARNING - invalid view handle]

SEE ALSO

DfInqViewGroup(3D), DoView(3D), DvInqDefinitionGroup(3D), DvUpdate(3D)

NAME

DvInqRendStyle – Return the rendering style of a view

SYNOPSIS

C:

```
DtRenderStyle DvInqRendStyle(view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQRS(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqRendStyle returns the rendering style of the view specified by the parameter *view*. The rendering style is a top level selection of the rendering process. There are two currently supported rendering styles:

DcRealTime <DCRLTM>
: Fast, interactive display rendering. This renderer runs fast enough for dynamic operations that require continuous updates of the scene without the user's awareness of rendering time.

DcProductionTime <DCPRTM>
: Slowest, most realistic rendering. This renderer does the most sophisticated rendering within the user's requested parameters. Doré may economize if the application's parameters negate the effect of the time-intensive techniques.

The value returned by *DvInqRendStyle* is equivalent to the current renderer identifier. Using a renderer other than the Dynamic Renderer or the Standard Production Renderer will result in a different renderer identifier returned from *DvInqRendStyle*.

ERRORS

DvInqRendStyle will fail if the view parameter is not a valid view; the returned value is undefined.

[WARNING - invalid view handle]

SEE ALSO

DvSetRendStyle(3D)

NAME

DvInqShadeIndex – Return the shade index of a view

SYNOPSIS

C:

```
DtInt DvInqShadeIndex (view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQSI(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqShadeIndex returns the shade index for the view specified by the parameter *view*. The shade index specifies the shade range index to use when converting the color attribute values of the view for display on a device with a visual type of *DcPseudoColor* <*DCPSUC*> and shade mode of *DcRange* <*DCRNG*>.

ERRORS

DvInqShadeIndex fails if the view handle is invalid.

[WARNING - invalid view handle]

SEE ALSO

DoShadeIndex(3D), DdInqShadeMode(3D), DdInqVisualType(3D), DvSetShadeIndex(3D)

NAME

DvInqUpdateType – Return the updatetype of a view

SYNOPSIS

C:

```
DtUpdateType DvInqUpdateType(view)
DtObject view;
```

Fortran:

```
INTEGER*4 DVQUT(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvInqUpdateType returns the update type of the view specified by the parameter *view*.

ERRORS

DvInqUpdateType will fail if the view handle is invalid; the returned value is undefined.

[WARNING - invalid view handle]

SEE ALSO

DvSetUpdateType(3D)

NAME

DvSetActiveCamera – Set the active camera for a view

SYNOPSIS

C:

```
void DvSetActiveCamera(view, camera)
DtObject view;
DtObject camera;
```

Fortran:

```
CALL DVSAC(VIEW, CAMERA)
INTEGER*4 VIEW
INTEGER*4 CAMERA
```

DESCRIPTION

DvSetActiveCamera sets the active camera to be used with the view specified by the parameter *view* when rendering. The active camera specifies which camera's attributes will be used to render a view. If this function is not called, the last camera defined in the studio definition group will be used as the active camera.

ERRORS

DvSetActiveCamera will fail if the view handle is invalid.

[WARNING - invalid view handle]

DvSetActiveCamera will fail if the camera handle is invalid.

[WARNING - invalid camera handle]

SEE ALSO

DvInqActiveCamera(3D)

NAME

DvSetBackgroundColor – Set the background color of a view

SYNOPSIS

C:

```
void DvSetBackgroundColor(view, colormodel, color)
DtObject view;
DtColorModel colormodel;
DtReal color[];
```

Fortran:

```
CALL DVSBC(VIEW, COLMOD, COLOR)
INTEGER*4 VIEW
INTEGER*4 COLMOD
REAL*8 COLOR(*)
```

DESCRIPTION

DvSetBackgroundColor sets the background color of the view, *view*. The parameter *colormodel* specifies the color model used. The parameter *color* specifies the color to which a view is cleared if the clear flag is set for the view, *view*, and the view is updated.

ERRORS

DvSetBackgroundColor will fail if the view handle is invalid.

[WARNING - invalid view handle]

DEFAULTS

The default background color is (*DcRGB*, 0.0, 0.0, 0.0).

SEE ALSO

DvInqBackgroundColor(3D)

NAME

DvSetBoundary – Set the view boundary

SYNOPSIS

C:

```
void DvSetBoundary(view, boundary)
DtObject view;
DtVolume *boundary;
```

Fortran:

```
CALL DVSB(VIEW, BNDRY)
INTEGER*4 VIEW
REAL*8 BNDRY(6)
```

DESCRIPTION

DvSetBoundary sets the view boundary for the view specified by the parameter *view*. The view boundary parameter *boundary* specifies position and amount of space occupied by the view within a frame.

ERRORS

DvSetBoundary will fail if the view handle is invalid.

[WARNING - invalid view handle]

DEFAULTS

The default view boundary extends from (0.0, 0.0, 0.0) to (1.0, 1.0, 1.0) in frame coordinates.

SEE ALSO

DvInqBoundary(3D)

NAME

DvSetClearFlag – Set the clear flag of a view

SYNOPSIS

C:

```
void DvSetClearFlag(view, clearflag)
DtObject view;
DtFlag clearflag;
```

Fortran:

```
CALL DVSCF(VIEW, CLRFLG)
INTEGER*4 VIEW
INTEGER*4 CLRFLG
```

DESCRIPTION

DvSetClearFlag sets the clear flag of a view specified by the parameter *view*. If the parameter *clearflag* is *DcTrue* <*DCTRUE*>, the view is first cleared to its background color every time the view is updated. If the *clearflag* is *DcFalse* <*DCFALS*>, the view is not cleared.

ERRORS

DvSetClearFlag will fail if the view handle is invalid.

[WARNING - invalid view handle]

DEFAULTS

The default clear flag is *DcTrue*.

SEE ALSO

DvInqClearFlag(3D)

NAME

DvSetRendStyle – Set the rendering style of a view

SYNOPSIS

C:

```
void DvSetRendStyle(view, renderstyle)
DtObject view;
DtRenderStyle renderstyle;
```

Fortran:

```
CALL DVSRS(VIEW, RNDSTL)
INTEGER*4 VIEW
INTEGER*4 RNDSTL
```

DESCRIPTION

DvSetRendStyle sets the rendering style of the view specified by the parameter *view*. The parameter *renderstyle* is a top level selection of the rendering process. There are two currently supported rendering styles:

DcRealTime <DCRLTM>
: Fast, interactive display rendering. This renderer runs fast enough for dynamic operations that require continuous updates of the scene without the user's awareness of rendering time.

DcProductionTime <DCPRTM>
: Slowest, most realistic rendering. This renderer does the most sophisticated rendering within the user's requested parameters. Doré may economize if the application's parameters negate the effect of the time-intensive techniques.

The actual techniques used by each of these *renderstyles* may change with new versions of Doré and with different hardware implementations; the trade-off of time and realism should remain the same.

The value of *renderstyle* is a renderer identifier. Using a renderer identifier other than the Dynamic Renderer identifier and the Standard Production Renderer identifier will select a new renderer (provided the renderer has been installed into your Doré configuration).

ERRORS

DvSetRendStyle will fail if the view is invalid.

[WARNING - invalid view handle]

DEFAULTS

The default render style is *DcRealTime <DCPRTM>*.

SEE ALSO

DvInqRendStyle(3D)

NAME

DvSetShadeIndex – Set the shade index of a view

SYNOPSIS

C:

```
void DvSetShadeIndex (view, index)
DtObject view;
DtInt index;
```

Fortran:

```
CALL DVSSI(VIEW, INDEX)
INTEGER*4 VIEW
INTEGER*4 INDEX
```

DESCRIPTION

DvSetShadeIndex sets the shade index for the view specified by the parameter *view*. The shade index parameter *index* specifies the shade range index to use when displaying on a device with a visual type of *DcPseudoColor* <*DCPSUC*> and a shade mode of *DcRange* <*DCRNG*>.

ERRORS

DvSetShadeIndex fails if the view handle is invalid.

[WARNING - invalid view handle]

DEFAULTS

The default *index* is 1.

SEE ALSO

DdInqShadeMode(3D), DdInqVisualType(3D), DoShadeIndex(3D), DvInqShadeIndex(3D)

NAME

DvSetUpdateType – Set the updatetype of a view

SYNOPSIS

C:

```
void DvSetUpdateType(view, updatetype)
DtObject view;
DtUpdateType updatetype;
```

Fortran:

```
CALL DVSUT(VIEW,UPDTYP)
INTEGER*4 VIEW
INTEGER*4 UPDTYP
```

DESCRIPTION

DvSetUpdateType sets the updatetype value for a view on subsequent updates. The parameter *updatetype* determines which elements of a *view* will be taken into account during the update. Possible values are:

DcUpdateAll <DCUALL>
: All objects, including cameras, lights, and display objects will be updated. This mode should be used if the cameras and lights have been modified since the last update of this view.

DcUpdateDisplay <DCUDIS>
: This mode causes only display objects to be updated. Camera and light information is the same as on previous updates.

ERRORS

DvSetUpdateType will fail if the view handle is invalid.

[WARNING - invalid view handle]

DEFAULTS

The default *updatetype* is *DcUpdateAll <DCUALL>*.

SEE ALSO

DdUpdate(3D), DfUpdate(3D), DvSetClearFlag(3D), DvSetRendStyle(3D)

NAME

DvUpdate – Redisplay the specified view

SYNOPSIS

C:

```
void DvUpdate(view)
DtObject view;
```

Fortran:

```
CALL DVU(VIEW)
INTEGER*4 VIEW
```

DESCRIPTION

DvUpdate causes the view, *view*, to update itself. Depending on the update type (set by *DvSetUpdateType <DVSUT>*), *DvUpdate* will update only display objects within the view, or display objects and studio objects within the view.

ERRORS

DvUpdate will fail if the view handle is invalid.

[WARNING - invalid view handle]

DvUpdate will fail if Doré is already performing a traversal of the database.

[WARNING - traversal already in progress]

SEE ALSO

DdUpdate(3D), DfUpdate(3D), DvSetClearFlag(3D), DvSetRendStyle(3D)

C INCLUDE FILES

CHAPTER TWO

This chapter contains a listing of the Doré include files for use with application programs written in C. This information is taken directly from the *dore.h* header file. This header file must be included in the beginning of every Doré application using a *#include* statement. Function prototyping is forced by inclusion (a *#include* statement is used) of the *dore_proto.h* prototypes file in the *dore.h* header file. Both of these files are located in the */usr/include* directory.

Filename: dore.h

```
/************************************************************************
Copyright (C) 1989, by Stardent Computer Corp.
        All Rights Reserved
This program is a  trade secret of Stardent Computer Corp. and it is not
to be reproduced, published, disclosed  to others, copied, adapted,
distributed, or displayed without the prior authorization of Stardent
Computer Corp.  Licensee agrees to attach or embed this Notice on all
copies of the program, including partial copies or modified versions
thereof.
************************************************************************/
#ifndef DORE_H
#define DORE_H

/*
 * Dore is set up to allow DtReal (floating point numbers used by Dore) to
 * be defined as either double precision or single precision.
 *
 * If DORE_REAL_SINGLE is defined then DtReal will be float.
 * If DORE_REAL_DOUBLE is defined then DtReal will be double.
 *
 * Stardent 1500 (TITAN P2) machines will default to double precision.
 * Stardent 3000 (TITAN P3) machines will default to double precision.
 *
 * The default is single precision.
 */

#if defined(DORE_REAL_SINGLE)
        typedef float DtReal;
#else
# if defined (DORE_REAL_DOUBLE)
        typedef double DtReal;
# else
#  if defined (titan)
        typedef double DtReal;
#  else
        typedef float DtReal;
#  endif
# endif
#endif

#if defined(__STDC__)
# define DORE_FCN_PROTOTYPES
#else
# if defined(titan)
#  define DORE_FCN_PROTOTYPES
# endif
#endif

/************************************************************************/
```

```
/**                                                                    **/
/**  This file contains all the Dore user type and constant            **/
/**  declarations.                                                     **/
/**                                                                    **/
/************************************************************************/

typedef void *DtPtr;

typedef unsigned long Dt32Bits; /* must always be exactly 32 bits large */

typedef DtPtr DtObject;

typedef short DtShort;

typedef unsigned short DtUShort;

typedef unsigned char DtUChar;

typedef long DtInt;             /* must always be at least 32 bits large */

typedef unsigned long DtUInt;   /* must always be at least 32 bits large */

typedef DtReal DtRealCouple[2]; /*  Couple of reals. */
typedef DtReal DtRealTriple[3]; /*  Triple of reals.  */
typedef DtReal DtRealQuad[4];   /*  Quadruplet of reals.  */
typedef DtInt DtIntTriple[3];   /*  Triple of ints.  */

typedef DtReal DtPoint3[3];
typedef DtReal DtPoint4[4];
typedef struct { DtReal x,y;     } DtNPoint2;
typedef struct { DtReal x,y,z;   } DtNPoint3;
typedef struct { DtReal x,y,z,w; } DtNPoint4;

typedef DtReal DtVector3[3];
typedef struct { DtReal dx,dy,dz; } DtNVector3;

typedef DtReal DtMatrix4x4[4][4];

typedef DtReal DtColorRGB[3];
typedef struct { DtReal r,g,b; } DtNColor3;

typedef struct {
        DtReal ll[2];
        DtReal ur[2];
} DtArea;

typedef struct {
        DtNPoint2 ll;
        DtNPoint2 ur;
} DtNArea;

typedef struct {
        DtPoint3 bll;
        DtPoint3 fur;
} DtVolume;
```

```
typedef struct {
        DtReal width;
        DtReal height;
        DtReal depth;
} DtSize3;

typedef struct {
        DtPoint3  point;
        DtVector3 vector;
} DtHalfSpace;

typedef struct {
        DtReal LowerBound;
        DtReal UpperBound;
} DtInterval;

typedef struct {
        DtInt depth ;
        DtShort width, height ;
        DtUChar *bits ;
} DtPattern ;

typedef DtInt (*DtPFI)() ;
typedef DtPtr (*DtPFF)() ;

typedef struct object {
        DtShort info ;
        DtShort ref_count ;
        DtPtr   data ;          /* object's private data */
        DtPtr  *additional_data ;
} DtObjectStructure;            /* only used with user-defined objects */

/**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**/

typedef DtInt DtUpdateType;
#define DcUpdateAll             0
#define DcUpdateDisplay         1

typedef char DtFlag;
#define DcFalse  0
#define DcTrue   1

typedef char DtSwitch;
#define DcOff  0
#define DcOn   1

typedef char DtSurface;
#define DcSphere    0
#define DcCylinder  1
#define DcBox       2
#define DcCone      3

typedef char DtRepType;
#define DcPoints     0
#define DcWireframe  1
#define DcSurface    2
```

```
#define DcOutlines   3

typedef char DtInterpType;
#define DcConstantShade  0
#define DcVertexShade    1
#define DcSurfaceShade   2

typedef char DtAxis;
#define DcXAxis  0
#define DcYAxis  1
#define DcZAxis  2

typedef char DtMajorPlane;
#define DcXY  0
#define DcYZ  1
#define DcXZ  2

typedef char DtColorModel;
#define DcRGB  0

typedef char DtVisualType;
#define DcStaticGrey   0
#define DcGreyScale    1
#define DcStaticColor  2
#define DcPseudoColor  3
#define DcTrueColor    4
#define DcDirectColor  5

typedef char DtShadeMode;
#define DcRange        0
#define DcComponent    1

typedef char DtVertexType;
#define DcLoc         0
#define DcLocNrm      1
#define DcLocClr      2
#define DcLocNrmClr   3

typedef char DtCtrlPointType;
#define DcCtr     0
#define DcCtrClr  1

typedef char DtCompType;
#define DcPreConcatenate   0
#define DcPostConcatenate  1
#define DcReplace          2

typedef char DtRenderStyle;
#define DcRealTime         0
#define DcProductionTime   2

typedef char DtShapeType;
#define DcConvex     0
#define DcConcave    1
#define DcComplex    2
```

```
typedef char DtRelPosition;
#define DcBeginning  0
#define DcEnd        1
#define DcCurrent    2

typedef char DtProjectionType;
#define DcParallel     0
#define DcPerspective  1

typedef char DtCameraMatrixType;
#define DcCameraArbitrary       0
#define DcCameraParallel        1
#define DcCameraPerspective     2
#define DcCameraProjection      3

typedef char DtClipOperator;
#define DcClipAll           0
#define DcClipAnd           1
#define DcClipAndReverse    2
#define DcClipNoOp          3
#define DcClipAndInverted   4
#define DcClipReplace       5
#define DcClipXOr           6
#define DcClipOr            7
#define DcClipNor           8
#define DcClipEqv           9
#define DcClipInvertVolume 10
#define DcClipOrReverse    11
#define DcClipInvert       12
#define DcClipOrInverted   13
#define DcClipNAnd         14
#define DcClipNone         15

typedef char DtTextPrecision;
#define DcStringPrecision     0
#define DcCharacterPrecision  1
#define DcStrokePrecision     2

typedef char DtTextAlignHorizontal;
#define DcTextHAlignNormal  0
#define DcTextHAlignLeft    1
#define DcTextHAlignCenter  2
#define DcTextHAlignRight   3

typedef char DtTextAlignVertical;
#define DcTextVAlignNormal  0
#define DcTextVAlignTop     1
#define DcTextVAlignCap     2
#define DcTextVAlignHalf    3
#define DcTextVAlignBase    4
#define DcTextVAlignBottom  5

typedef char DtTextPath;
#define DcTextPathRight  0
#define DcTextPathLeft   1
#define DcTextPathUp     2
```

```
#define DcTextPathDown   3

typedef char DtNameType;
#define DcNameNone     0
#define DcNameInteger  1
#define DcNameString   2

typedef char DtPickPathOrder;
#define DcTopFirst     0
#define DcBottomFirst  1

typedef char DtHitStatus;
#define DcHitAccept       0
#define DcHitReject       1
#define DcHitOverwrite    2

typedef char DtFilter;
#define DcInvisibilityInclusion  0
#define DcInvisibilityExclusion  1
#define DcPickabilityInclusion   2
#define DcPickabilityExclusion   3

typedef char DtSetOperation;
#define DcSetAdd      0
#define DcSetDelete   1
#define DcSetInvert   2
#define DcSetReplace  3

typedef char DtLineType;
#define DcLineTypeSolid    0
#define DcLineTypeDash     1
#define DcLineTypeDot      2
#define DcLineTypeDotDash  3

typedef char DtGroupNetworkStatus;
#define DcGroupOk    0
#define DcGroupBad   1

typedef char DtErrorStatus;
#define DcErrorMinor    0
#define DcErrorSevere   1
#define DcErrorFatal    2

typedef char DtReadStatus;
#define DcReadOk    0
#define DcReadTrunc 1
#define DcReadUnsuc 2

typedef char DtObjectStatus;
#define DcObjectValid    0
#define DcObjectInvalid  1
#define DcObjectDeleted  2

typedef DtInt DtFont;
#define DcPlainRoman            0
#define DcSimplexRoman          1
```

```
#define DcDuplexRoman           2
#define DcComplexSmallRoman     3
#define DcComplexRoman          4
#define DcTriplexRoman          5
#define DcComplexSmallItalic    6
#define DcComplexItalic         7
#define DcTriplexItalic         8
#define DcSimplexScript         9
#define DcComplexScript         10
#define DcGothicGerman          11
#define DcGothicEnglish         12
#define DcGothicItalian         13
#define DcPlainGreek            14
#define DcSimplexGreek          15
#define DcComplexSmallGreek     16
#define DcComplexGreek          17
#define DcComplexCyrillic       18
#define DcUpperCaseMathematics  19
#define DcLowerCaseMathematics  20
#define DcMusic                 21
#define DcMeteorology           22
#define DcSymbols               23
#define DcAstrology             24
#define DcHelvetica             25

typedef struct {
        DtFont  font ;
        DtTextPrecision precision ;
} DtFontPrecision ;

typedef DtInt DtRasterType;
#define DcRasterRGB             0
#define DcRasterRGBA            1
#define DcRasterRGBAZ           2
#define DcRasterRGBZ            3
#define DcRasterA               4
#define DcRasterZ               5
#define DcRasterSpecial         99

typedef DtInt DtTextureAntiAliasMode;
#define DcTextureAntiAliasNone          0
#define DcTextureAntiAliasMIP           1
#define DcTextureAntiAliasAdaptive      2

typedef DtInt DtExtendMode;
#define DcExtendNone            0
#define DcExtendBlack           1
#define DcExtendClamp           2
#define DcExtendRepeat          3

typedef DtInt DtTextureOperator;
#define DcTextureReplace        0
#define DcTextureMultiply       1
#define DcTextureBlend          2
#define DcTextureAdd            3
```

```
typedef DtInt DtMapOperator;
#define DcMapReplace            0
#define DcMapAdd                1

/**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**/

#define DcNullPtr         (DtPtr)0

#define DcNullObject      (DtObject)0

/* subdivision types */
#define DcSubDivFixed               0
#define DcSubDivAbsolute            1
#define DcSubDivRelative            2
#define DcSubDivSegments            3

/* methods */
#define DcMethodNull                    -1
#define DcMethodDestroy                 0
#define DcMethodAddReference            1
#define DcMethodRemoveReference         2
#define DcMethodPrint                   3
#define DcMethodCheckGroup              4
#define DcMethodCmpBndVolume            5
#define DcMethodIniPick                 6
#define DcMethodPick                    7
#define DcMethodInqGlbAttVal            8
#define DcMethodUpdStdAltObj            9
#define DcMethodStdRenderStudio         10
#define DcMethodStdRenderDisplay        11

/* rendering methods */
#define DcMethodDynIniRender            12
#define DcMethodDynRender               13
#define DcMethodGlbrndIniRender         14
#define DcMethodGlbrndIniObjs           15
#define DcMethodGlbrndSpcboxOvr         16
#define DcMethodGlbrndRayint            17
#define DcMethodGlbrndUsrdat            18
#define DcMethodGlbrndWcsloc            19
#define DcMethodGlbrndWcsnrm            20
#define DcMethodGlbrndWldBnd            21

/* pick status */
#define DcPickBadStatus         1
#define DcPickListOverflow      2
#define DcPickIndexOverflow     4

/* marker types */
#define DcMarkerPoint           -1
#define DcMarkerPlus            1
#define DcMarkerStar            2
#define DcMarker0               3
#define DcMarkerX               4
#define DcMarkerDiamond         5
#define DcMarkerSquare          6
```

```
#define DcMarkerTriangle        7

/* object types */
/*************************************************************************
 ************************** WARNING ********************************
 ** Dore 2.2 and older use constants for identifying classes.          **
 ** The following constants of the form 'DcType...' have been          **
 ** retained for backward compatibility.                               **
 ** As new classes are added they will NOT have assigned constants.    **
 ** Instead the class identifier should be obtained from the class     **
 ** name through the system routine DsInqClassId.  The class name for  **
 ** all classes (both new and old) is the same as the name of the C    **
 ** routine used to create an instance of the class.  For example the  **
 ** class name for a View is "DoView", corresponding to the routine    **
 ** DoView().                                                          **
 *************************************************************************/

#define DcTypeAny                       -1
#define DcTypeClass                     0
#define DcTypeDeleted                   1

#define DcTypeGroup                     2
#define DcTypeInlineGroup               3
#define DcTypeDevice                    4
#define DcTypeFrame                     5
#define DcTypeView                      6
#define DcTypeLabel                     7
#define DcTypeTransformMatrix           8
#define DcTypeLookAtFrom                9
#define DcTypePopMatrix                 10
#define DcTypePushMatrix                11
#define DcTypeRotate                    12
#define DcTypeScale                     13
#define DcTypeShear                     14
#define DcTypeTranslate                 15
#define DcTypeAmbientSwitch             16
#define DcTypeAnnotationText            17
#define DcTypeBackfaceCullSwitch        20
#define DcTypeBoundingVolume            21
#define DcTypeCallback                  22
#define DcTypeCamera                    23
#define DcTypeCameraMatrix              25
#define DcTypeClipSwitch                26
#define DcTypeClipVolume                27
#define DcTypeDataPtr                   29
#define DcTypeDataVal                   30
#define DcTypeDepthCue                  31
#define DcTypeDepthCueSwitch            32
#define DcTypeDiffuseColor              33
#define DcTypeDiffuseIntens             34
#define DcTypeDiffuseSwitch             35
#define DcTypeExecSet                   37
#define DcTypeFilter                    38
#define DcTypeGlbRndMaxObjs             39
#define DcTypeGlbRndMaxSub              40
#define DcTypeGlbRndRayLevel            41
```

```
#define DcTypeHiddenSurfaceSwitch       42
#define DcTypeInputSlot                 45
#define DcTypeInterpType                46
#define DcTypeInvisibilitySwitch        47
#define DcTypeLight                     48
#define DcTypeLightColor                49
#define DcTypeLightIntens               52
#define DcTypeLightType                 55
#define DcTypeLineType                  57
#define DcTypeLineWidth                 58
#define DcTypeMatrix                    59
#define DcTypeMinBoundingExtension      60
#define DcTypeMarkerFont                61
#define DcTypeMarkerGlyph               62
#define DcTypeMarkerScale               63
#define DcTypeNameSet                   64
#define DcTypeNURBSurface               66
#define DcTypeParallel                  69
#define DcTypePatch                     70
#define DcTypePickID                    71
#define DcTypePickabilitySwitch         72
#define DcTypePerspective               73
#define DcTypePolygon                   74
#define DcTypePolyline                  75
#define DcTypePolymarker                76
#define DcTypePolygonMesh               77
#define DcTypePopAttributes             78
#define DcTypePrimitiveSurface          79
#define DcTypeProjection                80
#define DcTypePushAttributes            81
#define DcTypeReflectionSwitch          82
#define DcTypeRepType                   83
#define DcTypeShadowSwitch              84
#define DcTypeShadeIndex                85
#define DcTypeSimplePolygon             86
#define DcTypeSimplePolygonMesh         87
#define DcTypeSpecularColor             91
#define DcTypeSpecularFactor            92
#define DcTypeSpecularIntens            93
#define DcTypeSpecularSwitch            94
#define DcTypeTorus                     98
#define DcTypeTriangleMesh              100
#define DcTypeTranspColor               101
#define DcTypeTranspIntens              102
#define DcTypeTranspSwitch              103
#define DcTypeText                      104
#define DcTypeTextAlign                 105
#define DcTypeTextExpansionFactor       106
#define DcTypeTextFont                  107
#define DcTypeTextHeight                108
#define DcTypeTextPath                  109
#define DcTypeTextPrecision             110
#define DcTypeTextSpace                 111
#define DcTypeTextUpVector              112
#define DcTypeAttAmbientSwitch          115
#define DcTypeAttDiffuseSwitch          116
```

```
#define DcTypeAttSpecularSwitch          117
#define DcTypeAttDiffuseColor            118
#define DcTypeAttSpecularColor           119
#define DcTypeAttDiffuseIntens           120
#define DcTypeAttSpecularIntens          121
#define DcTypeAttSpecularFactor          122
#define DcTypeAttRepType                 123
#define DcTypeAttInterpType              124
#define DcTypeAttLcstowcsmat             125
#define DcTypeAttBackfaceCullSwitch      126
#define DcTypeBackfaceCullable           127
#define DcTypeAttBackfaceCullable        128
#define DcTypeAttLightIntens             129
#define DcTypeAttLightColor              131
#define DcTypeAttShadowSwitch            132
#define DcTypeAttReflectionSwitch        133
#define DcTypeAttGlbRndMaxObjs           134
#define DcTypeAttGlbRndMaxSub            135
#define DcTypeAttGlbRndRayLevel          136
#define DcTypeAttTranspColor             137
#define DcTypeAttTranspIntens            138
#define DcTypeAttTranspSwitch            139
#define DcTypeAttTextAlignment           145
#define DcTypeAttTextExpansion           146
#define DcTypeAttTextFont                147
#define DcTypeAttTextHeight              148
#define DcTypeAttTextPath                149
#define DcTypeAttTextSpace               150
#define DcTypeAttTextUpVector            151
#define DcTypeAttTextPrecision           152
#define DcTypeRayinttri                  153
#define DcTypeRayintmshtri               154
#define DcTypeNull                       155
#define DcTypeAttLineType                156
#define DcTypeAttLineWidth               157
#define DcTypeAttMarkerScale             158
#define DcTypeAttMarkerGlyph             159
#define DcTypeAttMarkerFont              160
#define DcTypeAttExecSet                 161
#define DcTypeAttNameSet                 162
#define DcTypeAttFilter                  163
#define DcTypeAttInvisibilitySwitch      164
#define DcTypeAttPickabilitySwitch       165
#define DcTypeAttClipSwitch              168
#define DcTypeAttClipVolume              169
#define DcTypeBoundingVolumeSwitch       170
#define DcTypeAttBoundingVolumeSwitch    171
#define DcTypeAttMinBoundingExtension    172
#define DcTypeAttPickID                  173
#define DcTypeTriangleList               175
#define DcTypeAttDepthCue                176
#define DcTypeAttDepthCueSwitch          177
#define DcTypeAttHiddenSurfaceSwitch     178
#define DcTypeSubDivSpec                 179
#define DcTypeAttSubDivSpec              180
#define DcTypeAttLightType               182
```

```
#define DcTypeSurfaceShader             183
#define DcTypeAttSurfaceShader          184
#define DcTypeLineList                  185
#define DcTypePointList                 186
#define DcTypeAttShadeIndex             187
#define DcTypeSphereList                189
#define DcTypeCylinderList              190
#define DcTypeStereoSwitch              193
#define DcTypeAttStereoSwitch           194
#define DcTypeStereo                    195
#define DcTypeAttStereo                 196
#define DcTypeAmbientIntens             199
#define DcTypeAttAmbientIntens          200
#define DcTypeVarTriangleMesh           201
#define DcTypeVarLineList               202
#define DcTypeVarPointList              203
#define DcTypeVarSimplePolygonMesh      204
#define DcTypeAttCammat                 205

/**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**/

/* standard matrices for cubic curves and patches */

extern DtObject DcBezier4;
extern DtObject DcHermite4;
extern DtObject DcBSpline4;

/* standard light type objects */

extern DtObject DcLightAmbient;
extern DtObject DcLightInfinite;
extern DtObject DcLightPoint;
extern DtObject DcLightPointAttn;
extern DtObject DcLightSpot;
extern DtObject DcLightSpotAttn;

/* standard surface shader objects */

extern DtObject DcShaderConstant;
extern DtObject DcShaderLightSource;

/* standard pick callbacks */

extern DtObject DcPickFirst;
extern DtObject DcPickAll;
extern DtObject DcPickClosest;

/* standard input slots */

extern DtObject DcTransXSlot;
extern DtObject DcTransYSlot;
extern DtObject DcTransZSlot;
extern DtObject DcScaleXSlot;
extern DtObject DcScaleYSlot;
extern DtObject DcScaleZSlot;
extern DtObject DcRotXSlot;
```

```
extern DtObject DcRotYSlot;
extern DtObject DcRotZSlot;
extern DtObject DcUndoSlot;
extern DtObject DcUpdateSlot;

/* standard Texture Map Operators */

extern DtObject DcStdBumpMap;
extern DtObject DcStdSphereEnvironMap;
extern DtObject DcStdCubeEnvironMap;
extern DtObject DcStdTableLookup;
extern DtObject DcStd3dTableLookup;

/* standard Super Sample Filter */

extern DtObject DcFilterBox;

/* standard callback to delete raster data */

extern DtObject DcDeleteData;

/* include the Dore function prototypes */

#include <dore_proto.h>

#endif
/* end of dore.h */
```

Filename: dore_proto.h

```
/**************************************************************************
Copyright (C) 1989, by Stardent Computer Corp.
        All Rights Reserved
This program is a  trade secret of Stardent Computer Corp. and it is not
to be reproduced, published, disclosed to others, copied, adapted,
distributed, or displayed without the prior authorization of Stardent
Computer Corp.  Licensee agrees to attach or embed this Notice on all
copies of the program, including partial copies or modified versions
thereof.
**************************************************************************/
#ifndef DORE_PROTO_H
#define DORE_PROTO_H
/*
 * File: dore_proto.h
 *
 * This file contains prototype declarations for user-visible dore
 * functions.
 *
 */

#if defined(DORE_FCN_PROTOTYPES)
#define ARGS(arg_list) arg_list
#else
#define ARGS(arg_list) ( )
#endif

/*
 * Some parameters look different from the user and implementation sides.
 * This stuff allows users to use either 2D or 1D arrays in certain places.
 */
#if defined(dod_address_null)
#define PROTO_AREA              DtNArea
#define PROTO_INT_LIST          DtInt  []
#define PROTO_REAL_ARRAY        DtReal [][3]
#define PROTO_REAL_LIST         DtReal []
#else
#define PROTO_AREA              DtArea
#define PROTO_INT_LIST          void *
#define PROTO_REAL_ARRAY        void *
#define PROTO_REAL_LIST         void *
#endif


/*
 * Derived types from DtObject (for stronger typing).
 * Devices, frames, groups, and views are Dore objects and
 * may be used anywhere a DtObject is used; but a DtObject
 * of the wrong type may not be used where a more specific
 * type is called for.
```

```
 */
typedef DtObject        DtDevice;               /* Dd functions */
typedef DtObject        DtFrame;                /* Df functions */
typedef DtObject        DtGroup;                /* Dg functions */
typedef DtObject        DtView;                 /* Dv functions */


/*
 * Device Functions
 */
void            DdInqColorEntries       ARGS( (DtDevice, DtColorModel,
                                               DtInt, DtInt, DtReal[]) );
DtInt           DdInqColorTableSize     ARGS( (DtDevice) );
void            DdInqExtent             ARGS( (DtDevice, DtVolume *) );
void            DdInqFonts              ARGS( (DtDevice, DtFontPrecision[]) );
DtFrame         DdInqFrame              ARGS( (DtDevice) );
DtInt           DdInqNumFonts           ARGS( (DtDevice) );
void            DdInqPickAperture       ARGS( (DtDevice, DtSize3 *) );
DtObject        DdInqPickCallback       ARGS( (DtDevice) );
DtPickPathOrder DdInqPickPathOrder      ARGS( (DtDevice) );
DtFlag          DdInqPixelData          ARGS( (DtDevice, DtRasterType, DtInt *,
                                               DtInt *, DtRasterType *, DtPtr *,
                                               DtFlag *) );
void            DdInqResolution         ARGS( (DtDevice, DtReal *, DtReal *) );
DtShadeMode     DdInqShadeMode          ARGS( (DtDevice) );
void            DdInqShadeRanges        ARGS( (DtDevice, DtInt, DtInt,
                                               DtInt[]) );
void            DdInqViewport           ARGS( (DtDevice, DtVolume *) );
DtVisualType    DdInqVisualType         ARGS( (DtDevice) );
void            DdPickObjs              ARGS( (DtDevice, DtPoint3, DtInt *,
                                               DtInt, DtInt[], DtInt, DtInt[],
                                               DtReal[], DtReal[], DtReal[],
                                               DtView[], DtInt *) );
/* NOTE: DdPick is obseleted by DdPickObjs */
void            DdPick                  ARGS( (DtDevice, DtPoint3, DtInt *,
                                               DtInt, DtInt[], DtInt, DtInt[],
                                               DtReal[], DtView[], DtInt *) );
void            DdSetColorEntries       ARGS( (DtDevice, DtColorModel,
                                               DtInt, DtInt,
                                               PROTO_REAL_LIST) );
void            DdSetFrame              ARGS( (DtDevice, DtFrame) );
void            DdSetPickAperture       ARGS( (DtDevice, DtSize3 *) );
void            DdSetPickCallback       ARGS( (DtDevice, DtObject) );
void            DdSetPickPathOrder      ARGS( (DtDevice, DtPickPathOrder) );
void            DdSetShadeMode          ARGS( (DtDevice, DtShadeMode) );
void            DdSetShadeRanges        ARGS( (DtDevice, DtInt, DtInt,
                                               DtInt[]) );
void            DdSetViewport           ARGS( (DtDevice, DtVolume *) );
void            DdUpdate                ARGS( (DtDevice) );


/*
 * Frame Functions
 */
void            DfInqBoundary           ARGS( (DtFrame, DtVolume *) );
void            DfInqJust               ARGS( (DtFrame, DtReal *, DtReal *) );
```

```
DtGroup             DfInqViewGroup              ARGS( (DtFrame) );
void                DfSetBoundary               ARGS( (DtFrame, DtVolume *) );
void                DfSetJust                   ARGS( (DtFrame, DtReal, DtReal) );
void                DfUpdate                    ARGS( (DtFrame) );


/*
 * Group Functions
 */
void                DgAddObj                    ARGS( (DtObject) );
void                DgAddObjToGroup             ARGS( (DtGroup, DtObject) );
DtGroupNetworkStatus DgCheck                    ARGS( (DtGroup) );
DtGroup             DgClose                     ARGS( (void) );
void                DgDelEle                    ARGS( (DtInt) );
DtFlag              DgDelEleBetweenLabels       ARGS( (DtInt, DtInt) );
void                DgDelEleRange               ARGS( (DtInt, DtInt) );
void                DgEmpty                     ARGS( (DtGroup) );
DtInt               DgInqElePtr                 ARGS( (void) );
DtObject            DgInqObjAtPos               ARGS( (DtGroup, DtInt,
                                                       DtRelPosition) );
DtGroup             DgInqOpen                   ARGS( (void) );
DtInt               DgInqSize                   ARGS( (DtGroup) );
void                DgOpen                      ARGS( (DtGroup, DtFlag) );
void                DgReplaceObj                ARGS( (DtObject) );
void                DgReplaceObjInGroup         ARGS( (DtGroup, DtObject) );
void                DgSetElePtr                 ARGS( (DtInt, DtRelPosition) );
DtFlag              DgSetElePtrRelLabel         ARGS( (DtInt, DtInt) );


/*
 * Object Creation Functions
 */
DtObject            DoAmbientIntens             ARGS( (DtReal) );
DtObject            DoAmbientSwitch             ARGS( (DtSwitch) );
DtObject            DoAnnoText                  ARGS( (DtReal[], DtPtr) );
DtObject            DoBackfaceCullSwitch        ARGS( (DtSwitch) );
DtObject            DoBackfaceCullable          ARGS( (DtSwitch) );
DtObject            DoBoundingVol               ARGS( (DtVolume *, DtObject) );
DtObject            DoBoundingVolSwitch         ARGS( (DtSwitch) );
DtObject            DoCallback                  ARGS( (DtPtr, DtObject) );
DtObject            DoCamera                    ARGS( (void) );
DtObject            DoCameraMatrix              ARGS( (DtMatrix4x4) );
DtObject            DoClipSwitch                ARGS( (DtSwitch) );
DtObject            DoClipVol                   ARGS( (DtClipOperator, DtInt,
                                                       DtHalfSpace[]) );
DtObject            DoDataPtr                   ARGS( (DtPtr) );
DtObject            DoDataVal                   ARGS( (Dt32Bits) );
DtObject            DoDepthCue                  ARGS( (DtReal, DtReal, DtReal, DtReal,
                                                       DtColorModel, DtReal[]) );
DtObject            DoDepthCueSwitch            ARGS( (DtSwitch) );
DtDevice            DoDevice                    ARGS( (DtPtr, DtPtr) );
DtObject            DoDiffuseColor              ARGS( (DtColorModel, DtReal[]) );
DtObject            DoDiffuseIntens             ARGS( (DtReal) );
DtObject            DoDiffuseSwitch             ARGS( (DtSwitch) );
DtObject            DoExecSet                   ARGS( (DtInt, DtInt[],
                                                       DtSetOperation) );
```

```
DtObject        DoFileRaster            ARGS( (DtPtr, DtPtr) );
DtObject        DoFilter                ARGS( (DtFilter, DtInt, DtInt[],
                                              DtSetOperation) );
DtFrame         DoFrame                 ARGS( (void) );
DtObject        DoGenerateTextureUV     ARGS( (DtSwitch) );
DtObject        DoGlbRendMaxObjs        ARGS( (DtInt) );
DtObject        DoGlbRendMaxSub         ARGS( (DtInt) );
DtObject        DoGlbRendRayLevel       ARGS( (DtInt) );
DtGroup         DoGroup                 ARGS( (DtFlag) );
DtObject        DoHiddenSurfSwitch      ARGS( (DtSwitch) );
DtObject        DoInLineGroup           ARGS( (DtFlag) );
DtObject        DoInputSlot             ARGS( (void) );
DtObject        DoInterpType            ARGS( (DtInterpType) );
DtObject        DoInvisSwitch           ARGS( (DtSwitch) );
DtObject        DoLabel                 ARGS( (DtInt) );
DtObject        DoLight                 ARGS( (void) );
DtObject        DoLightSpreadAngles     ARGS( (DtReal, DtReal) );
DtObject        DoLightSpreadExp        ARGS( (DtReal) );
DtObject        DoLightAttenuation      ARGS( (DtReal, DtReal) );
DtObject        DoLightSwitch           ARGS( (DtObject, DtSwitch) );
DtObject        DoLightColor            ARGS( (DtColorModel, DtReal[]) );
DtObject        DoLightIntens           ARGS( (DtReal) );
DtObject        DoLightType             ARGS( (DtObject) );
DtObject        DoLineList              ARGS( (DtColorModel, DtVertexType,
                                              DtInt, PROTO_REAL_LIST) );
DtObject        DoLineType              ARGS( (DtLineType) );
DtObject        DoLineWidth             ARGS( (DtReal) );
DtObject        DoLookAtFrom            ARGS( (DtPoint3, DtPoint3,
                                              DtVector3) );
DtObject        DoMarkerFont            ARGS( (DtFont) );
DtObject        DoMarkerGlyph           ARGS( (DtInt) );
DtObject        DoMarkerScale           ARGS( (DtReal) );
DtObject        DoMatrix                ARGS( (DtInt, DtInt,
                                              PROTO_REAL_LIST) );
DtObject        DoMinBoundingVolExt     ARGS( (DtReal) );
DtObject        DoNURBSurf              ARGS( (DtColorModel, DtCtrlPointType,
                                              PROTO_AREA,
                                              DtInt, DtInt, PROTO_REAL_LIST,
                                              DtInt, DtInt, PROTO_REAL_LIST,
                                              DtInt, DtInt, PROTO_REAL_LIST
                                              ) );
DtObject        DoNameSet               ARGS( (DtInt, DtInt[],
                                              DtSetOperation) );
DtObject        DoParallel              ARGS( (DtReal, DtReal, DtReal) );
DtObject        DoPatch                 ARGS( (DtColorModel, DtVertexType,
                                              DtObject, PROTO_REAL_LIST,
                                              DtObject) );
DtObject        DoPerspective           ARGS( (DtReal, DtReal, DtReal) );
DtObject        DoPickID                ARGS( (DtInt) );
DtObject        DoPickSwitch            ARGS( (DtSwitch) );
DtObject        DoPointList             ARGS( (DtColorModel, DtVertexType,
                                              DtInt, PROTO_REAL_LIST) );
DtObject        DoPolygon               ARGS( (DtColorModel, DtVertexType,
                                              DtInt, DtInt[],
                                              PROTO_REAL_LIST,
                                              DtShapeType) );
```

```
DtObject        DoPolygonMesh           ARGS( (DtColorModel, DtVertexType,
                                              DtInt, PROTO_REAL_LIST,
                                              DtInt, DtInt[],
                                              DtInt[], DtInt[],
                                              DtShapeType, DtFlag) );
DtObject        DoPolyline              ARGS( (DtColorModel, DtVertexType,
                                              DtInt, PROTO_REAL_LIST) );
DtObject        DoPolymarker            ARGS( (DtInt, PROTO_REAL_ARRAY) );
DtObject        DoPopAtts               ARGS( (void) );
DtObject        DoPopMatrix             ARGS( (void) );
DtObject        DoPrimSurf              ARGS( (DtSurface) );
DtObject        DoProjection            ARGS( (DtArea *, DtProjectionType,
                                              DtPoint3, DtReal,
                                              DtReal, DtReal) );
DtObject        DoPushAtts              ARGS( (void) );
DtObject        DoPushMatrix            ARGS( (void) );
DtObject        DoRaster                ARGS( (DtInt, DtInt, DtInt, DtRasterType,
                                              DtPtr, DtPtr, DtObject) );
DtObject        DoReflectionSwitch      ARGS( (DtSwitch) );
DtObject        DoRefractionSwitch      ARGS( (DtSwitch) );
DtObject        DoRefractionIndex       ARGS( (DtReal) );
DtObject        DoRepType               ARGS( (DtRepType) );
DtObject        DoRotate                ARGS( (DtAxis, DtReal) );
DtObject        DoSampleAdaptive        ARGS( (DtReal) );
DtObject        DoSampleAdaptiveSwitch  ARGS( (DtSwitch) );
DtObject        DoSampleFilter          ARGS( (DtObject, DtReal, DtReal) );
DtObject        DoSampleJitter          ARGS( (DtReal) );
DtObject        DoSampleJitterSwitch    ARGS( (DtSwitch) );
DtObject        DoSampleSuper           ARGS( (DtInt, DtInt) );
DtObject        DoSampleSuperSwitch     ARGS( (DtSwitch) );
DtObject        DoScale                 ARGS( (DtReal, DtReal, DtReal) );
DtObject        DoShadeIndex            ARGS( (DtInt) );
DtObject        DoShadowSwitch          ARGS( (DtSwitch) );
DtObject        DoShear                 ARGS( (DtMajorPlane,
                                              DtReal, DtReal) );
DtObject        DoSimplePolygon         ARGS( (DtColorModel, DtVertexType,
                                              DtInt, PROTO_REAL_LIST,
                                              DtShapeType ) );
DtObject        DoSimplePolygonMesh     ARGS( (DtColorModel, DtVertexType,
                                              DtInt, PROTO_REAL_LIST,
                                              DtInt, DtInt[], DtInt[],
                                              DtShapeType, DtFlag) );
DtObject        DoSpecularColor         ARGS( (DtColorModel, DtReal[]) );
DtObject        DoSpecularFactor        ARGS( (DtReal) );
DtObject        DoSpecularIntens        ARGS( (DtReal) );
DtObject        DoSpecularSwitch        ARGS( (DtSwitch) );
DtObject        DoStereo                ARGS( (DtReal, DtReal) );
DtObject        DoStereoSwitch          ARGS( (DtSwitch) );
DtObject        DoSubDivSpec            ARGS( (DtInt, DtReal[]) );
DtObject        DoSurfaceShade          ARGS( (DtObject) );
DtObject        DoText                  ARGS( (DtPoint3,
                                              DtVector3, DtVector3, DtPtr) );
DtObject        DoTextAlign             ARGS( (DtTextAlignHorizontal,
                                              DtTextAlignVertical) );
DtObject        DoTextExpFactor         ARGS( (DtReal) );
DtObject        DoTextFont              ARGS( (DtFont) );
```

```
DtObject        DoTextHeight            ARGS( (DtReal) );
DtObject        DoTextPath              ARGS( (DtTextPath) );
DtObject        DoTextPrecision         ARGS( (DtTextPrecision) );
DtObject        DoTextSpace             ARGS( (DtReal) );
DtObject        DoTextUpVector          ARGS( (DtReal, DtReal) );
DtObject        DoTextureAntiAlias      ARGS( (DtTextureAntiAliasMode) );
DtObject        DoTextureExtendUV       ARGS( (DtExtendMode, DtExtendMode) );
DtObject        DoTextureExtendUVW      ARGS( (DtExtendMode, DtExtendMode,
                                               DtExtendMode) );
DtObject        DoTextureMapBump        ARGS( (DtMapOperator, DtObject,
                                               DtObject) );
DtObject        DoTextureMapBumpSwitch  ARGS( (DtSwitch) );
DtObject        DoTextureMapDiffuseColor
                                        ARGS( (DtMapOperator, DtObject,
                                               DtObject) );
DtObject        DoTextureMapDiffuseColorSwitch
                                        ARGS( (DtSwitch) );
DtObject        DoTextureMapEnviron     ARGS( (DtMapOperator, DtObject,
                                               DtObject) );
DtObject        DoTextureMapEnvironSwitch
                                        ARGS( (DtSwitch) );
DtObject        DoTextureMapTranspIntens
                                        ARGS( (DtMapOperator, DtObject,
                                               DtObject) );
DtObject        DoTextureMapTranspIntensSwitch
                                        ARGS( (DtSwitch) );
DtObject        DoTextureOp             ARGS( (DtTextureOperator) );
DtObject        DoTextureScaleUV        ARGS( (DtReal, DtReal) );
DtObject        DoTextureScaleUVW       ARGS( (DtReal, DtReal, DtReal) );
DtObject        DoTextureTranslateUV    ARGS( (DtReal, DtReal) );
DtObject        DoTextureTranslateUVW   ARGS( (DtReal, DtReal, DtReal) );
DtObject        DoTextureUVIndex        ARGS( (DtInt) );
DtObject        DoTextureUVWIndex       ARGS( (DtInt) );
DtObject        DoTorus                 ARGS( (DtReal, DtReal) );
DtObject        DoTransformMatrix       ARGS( (DtMatrix4x4, DtCompType) );
DtObject        DoTranslate             ARGS( (DtReal, DtReal, DtReal) );
DtObject        DoTranspColor           ARGS( (DtColorModel, DtReal[]) );
DtObject        DoTranspIntens          ARGS( (DtReal) );
DtObject        DoTranspSwitch          ARGS( (DtSwitch) );
DtObject        DoTranspOrientColor     ARGS( (DtColorModel, DtReal[]) );
DtObject        DoTranspOrientIntens    ARGS( (DtReal) );
DtObject        DoTranspOrientExp       ARGS( (DtReal) );
DtObject        DoTranspOrientSwitch    ARGS( (DtSwitch) );
DtObject        DoTriangleList          ARGS( (DtColorModel, DtVertexType,
                                               DtInt, PROTO_REAL_LIST) );
DtObject        DoTriangleMesh          ARGS( (DtColorModel, DtVertexType,
                                               DtInt, PROTO_REAL_LIST,
                                               DtInt, PROTO_INT_LIST,
                                               DtFlag) );
DtObject        DoVarTriangleMesh       ARGS( (DtColorModel, DtInt,
                                               PROTO_REAL_LIST,
                                               PROTO_REAL_LIST,
                                               PROTO_REAL_LIST,
                                               DtInt, PROTO_INT_LIST,
                                               DtFlag) );
DtObject        DoVarSimplePolygonMesh  ARGS( (DtColorModel, DtInt,
```

```
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                DtInt, PROTO_INT_LIST,
                                                PROTO_INT_LIST,
                                                DtShapeType, DtFlag) );
DtObject        DoVarLineList           ARGS( (DtColorModel, DtInt,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST) );
DtObject        DoVarPointList          ARGS( (DtColorModel, DtInt,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST) );
DtView          DoView                  ARGS( (void) );

/*
 * Primitive Object Functions
 */
void            DpUpdVarTriangleMesh    ARGS( (DtObject,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                DtFlag) );
void            DpUpdVarSimplePolygonMesh ARGS( (DtObject,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                DtShapeType, DtFlag,
                                                DtFlag) );
void            DpUpdVarLineList        ARGS( (DtObject,
                                                DtInt,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST) );
void            DpUpdVarPointList       ARGS( (DtObject,
                                                DtInt,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST,
                                                PROTO_REAL_LIST) );

/*
 * System Functions
 */
DtVolume *      DsCompBoundingVol       ARGS( (DtVolume *, DtObject) );
void            DsExecuteObj            ARGS( (DtObject) );
void            DsExecutionAbort        ARGS( (void) );
void            DsExecutionReturn       ARGS( (void) );
DtInt           DsFileRasterRead        ARGS( (DtPtr, DtInt *, DtInt *,
                                                DtInt *, DtRasterType *,
                                                DtPtr *) );
DtObject        DsHoldObj               ARGS( (DtObject) );
void            DsInitializeSystem      ARGS( (DtInt) );
void            DsInputValue            ARGS( (DtObject, DtReal) );
DtInt           DsInqClassId            ARGS( (DtPtr) );
DtInt           DsInqCurrentMethod      ARGS( (void) );
```

```
DtReadStatus    DsInqErrorMessage       ARGS( (DtInt, DtInt, char[],
                                              DtErrorStatus *) );
void            DsInqErrorVars          ARGS( (DtInt *, DtPFI *) );
DtInt           DsInqExeDepthLimit      ARGS( (void) );
DtFlag          DsInqHoldObj            ARGS( (DtObject) );
DtInt           DsInqNumRenderers       ARGS( (void) );
DtInt           DsInqMethodId           ARGS( (DtPtr) );
DtObject        DsInqObj                ARGS( (DtNameType, DtPtr, DtInt) );
void            DsInqObjName            ARGS( (DtObject, DtNameType *,
                                              DtPtr) );
DtObjectStatus  DsInqObjStatus          ARGS( (DtObject) );
DtInt           DsInqObjType            ARGS( (DtObject) );
DtInt           DsInqRendererId         ARGS( (DtPtr) );
void            DsInqRendererNames      ARGS( (DtPtr []) );
DtFlag          DsInqSafeFlag           ARGS( (void) );
DtObject        DsInqValuatorGroup      ARGS( (DtObject) );
void            DsInqVersion            ARGS( (DtPtr *) );
void            DsPrintObj              ARGS( (DtObject) );
void            DsRasterUpdate          ARGS( (DtObject) );
DtInt           DsRasterWrite           ARGS( (DtObject, DtPtr) );
void            DsReleaseObj            ARGS( (DtObject) );
void            DsSetErrorVars          ARGS( (DtInt, DtPFI) );
void            DsSetExeDepthLimit      ARGS( (DtInt) );
void            DsSetObjName            ARGS( (DtObject, DtNameType, DtPtr,
                                              DtFlag) );
void            DsSetSafeFlag           ARGS( (DtFlag) );
void            DsTerminateSystem       ARGS( (void) );
DtVertexType    DsTextureUVCount        ARGS( (DtInt) );
DtVertexType    DsTextureUVWCount       ARGS( (DtInt) );
void            DsUpdateAllViews        ARGS( (void) );
void            DsValuatorSwitch        ARGS( (DtSwitch) );


/*
 * Viewport Functions
 */
DtObject        DvInqActiveCamera       ARGS( (DtView) );
void            DvInqBackgroundColor    ARGS( (DtView, DtColorModel *,
                                              DtReal[]) );
void            DvInqBoundary           ARGS( (DtView, DtVolume *) );
DtFlag          DvInqClearFlag          ARGS( (DtView) );
DtGroup         DvInqDefinitionGroup    ARGS( (DtView) );
DtGroup         DvInqDisplayGroup       ARGS( (DtView) );
DtRenderStyle   DvInqRendStyle          ARGS( (DtView) );
DtInt           DvInqShadeIndex         ARGS( (DtView) );
DtUpdateType    DvInqUpdateType         ARGS( (DtView) );
void            DvSetActiveCamera       ARGS( (DtView, DtObject) );
void            DvSetBackgroundColor    ARGS( (DtView, DtColorModel,
                                              DtReal[]) );
void            DvSetBoundary           ARGS( (DtView, DtVolume *) );
void            DvSetClearFlag          ARGS( (DtView, DtFlag) );
void            DvSetRendStyle          ARGS( (DtView, DtRenderStyle) );
void            DvSetShadeIndex         ARGS( (DtView, DtInt) );
void            DvSetUpdateType         ARGS( (DtView, DtUpdateType) );
void            DvUpdate                ARGS( (DtView) );
```

```
/*
 * User extension functions
 */
DtInt           DeAddClass              ARGS( (DtPtr, DtInt, DtInt[], DtPFI) );
DtObject        DeCreateObject          ARGS( (DtInt, DtPtr) );
void            DeDeleteObject          ARGS( (DtObject) );
void            DeExecuteAlternate      ARGS( (DtObject) );
void            DeInitializeObjPick     ARGS( (DtObject) );
DtFlag          DeInqPickable           ARGS( (DtInt) );
DtFlag          DeInqRenderable         ARGS( (DtInt) );

#endif
/* end dore_proto.h */
```

FORTRAN INCLUDE FILES

CHAPTER THREE

This chapter contains a listing of the Doré include files for use with application programs written in Fortran. These files define all of the Doré variables and functions and may be found in the */usr/include/fortran* directory.

Filename: DORETYPES

```
C**********************************************************************
CCopyright (C) 1989, by Stardent Computer Corp.
C        All Rights Reserved
CThis program is a  trade secret of  Stardent Computer Corp. and it is not
Cto be reproduced, published, disclosed  to others, copied, adapted,
Cdistributed, or displayed without the prior authorization of Stardent
CComputer Corp.  Licensee agrees  to  attach  or  embed  this  Notice on
Call copies  of the program, including partial copies or modified versions
Cthereof.
C**********************************************************************
C
C**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**
C**                                                                      **
C**  This file contains all the Dore FORTRAN type constant declarations.  **
C**                                                                      **
C*************************** WARNING ************************************
C**                                                                      **
C**       Dore 2.2 and older use constants for identifying classes.      **
C**       The following constants of the form 'DCT...' have been         **
C**       retained for backward compatibility.                           **
C**       As new classes are added they will NOT have assigned constants.  **
C**       Instead the class identifier should be obtained from the class   **
C**       name through the system routine DSQCI (DsInqClassId).          **
C**       The class name for all classes (both new and old) is the same as  **
C**       the name of the C routine used to create an instance of the class.**
C**       For example the class name for a View is "DoView", corresponding  **
C**       to the routine DoView().                                        **
C**********************************************************************

C object types
C       DcTypeAny
      INTEGER*4 DCTANY
      PARAMETER (DCTANY =-1)
C       DcTypeClass
      INTEGER*4 DCTCLS
      PARAMETER (DCTCLS =0)
C       DcTypeDeleted
      INTEGER*4 DCTDEL
      PARAMETER (DCTDEL =1)
C       DcTypeGroup
      INTEGER*4 DCTGRP
      PARAMETER (DCTGRP =2)
C       DcTypeInlineGroup
      INTEGER*4 DCTING
      PARAMETER (DCTING =3)
C       DcTypeDevice
      INTEGER*4 DCTDEV
      PARAMETER (DCTDEV =4)
C       DcTypeFrame
```

```
      INTEGER*4 DCTFRM
      PARAMETER (DCTFRM =5)
C       DcTypeView
      INTEGER*4 DCTVW
      PARAMETER (DCTVW =6)
C       DcTypeLabel
      INTEGER*4 DCTLBL
      PARAMETER (DCTLBL =7)
C       DcTypeTransformMatrix
      INTEGER*4 DCTTMX
      PARAMETER (DCTTMX =8)
C       DcTypeLookAtFrom
      INTEGER*4 DCTLAF
      PARAMETER (DCTLAF =9)
C       DcTypePopMatrix
      INTEGER*4 DCTPPM
      PARAMETER (DCTPPM =10)
C       DcTypePushMatrix
      INTEGER*4 DCTPSM
      PARAMETER (DCTPSM =11)
C       DcTypeRotate
      INTEGER*4 DCTROT
      PARAMETER (DCTROT =12)
C       DcTypeScale
      INTEGER*4 DCTSCL
      PARAMETER (DCTSCL =13)
C       DcTypeShear
      INTEGER*4 DCTSHR
      PARAMETER (DCTSHR =14)
C       DcTypeTranslate
      INTEGER*4 DCTXLT
      PARAMETER (DCTXLT =15)
C       DcTypeAmbientSwitch
      INTEGER*4 DCTAS
      PARAMETER (DCTAS =16)
C       DcTypeAnnotationText
      INTEGER*4 DCTANT
      PARAMETER (DCTANT =17)
C       DcTypeBackfaceCullSwitch
      INTEGER*4 DCTBCS
      PARAMETER (DCTBCS =20)
C       DcTypeBoundingVolume
      INTEGER*4 DCTBV
      PARAMETER (DCTBV =21)
C       DcTypeCallback
      INTEGER*4 DCTCB
      PARAMETER (DCTCB =22)
C       DcTypeCamera
      INTEGER*4 DCTCM
      PARAMETER (DCTCM =23)
C       DcTypeCameraMatrix
      INTEGER*4 DCTCMX
      PARAMETER (DCTCMX =25)
C       DcTypeClipSwitch
      INTEGER*4 DCTCS
      PARAMETER (DCTCS =26)
```

```
C       DcTypeClipVolume
      INTEGER*4 DCTCV
      PARAMETER (DCTCV =27)
C       DcTypeDataPtr
      INTEGER*4 DCTDPT
      PARAMETER (DCTDPT =29)
C       DcTypeDataVal
      INTEGER*4 DCTDVL
      PARAMETER (DCTDVL =30)
C       DcTypeDepthCue
      INTEGER*4 DCTDCU
      PARAMETER (DCTDCU =31)
C       DcTypeDepthCueSwitch
      INTEGER*4 DCTDCS
      PARAMETER (DCTDCS =32)
C       DcTypeDiffuseColor
      INTEGER*4 DCTDFC
      PARAMETER (DCTDFC =33)
C       DcTypeDiffuseIntens
      INTEGER*4 DCTDFI
      PARAMETER (DCTDFI =34)
C       DcTypeDiffuseSwitch
      INTEGER*4 DCTDFS
      PARAMETER (DCTDFS =35)
C       DcTypeExecSet
      INTEGER*4 DCTEXS
      PARAMETER (DCTEXS =37)
C       DcTypeFilter
      INTEGER*4 DCTFLT
      PARAMETER (DCTFLT =38)
C       DcTypeGlbRndMaxObjs
      INTEGER*4 DCTGMO
      PARAMETER (DCTGMO =39)
C       DcTypeGlbRndMaxSub
      INTEGER*4 DCTGMS
      PARAMETER (DCTGMS =40)
C       DcTypeGlbRndRayLevel
      INTEGER*4 DCTGRL
      PARAMETER (DCTGRL =41)
C       DcTypeHiddenSurfaceSwitch
      INTEGER*4 DCTHSS
      PARAMETER (DCTHSS =42)
C       DcTypeInputSlot
      INTEGER*4 DCTINS
      PARAMETER (DCTINS =45)
C       DcTypeInterpType
      INTEGER*4 DCTINT
      PARAMETER (DCTINT =46)
C       DcTypeInvisibilitySwitch
      INTEGER*4 DCTIVS
      PARAMETER (DCTIVS =47)
C       DcTypeLight
      INTEGER*4 DCTL
      PARAMETER (DCTL =48)
C       DcTypeLightColor
      INTEGER*4 DCTLC
```

```
      PARAMETER (DCTLC =49)
C       DcTypeLightIntens
      INTEGER*4 DCTLI
      PARAMETER (DCTLI =52)
C       DcTypeLightType
      INTEGER*4 DCTLT
      PARAMETER (DCTLT =55)
C       DcTypeLineType
      INTEGER*4 DCTLNT
      PARAMETER (DCTLNT =57)
C       DcTypeLineWidth
      INTEGER*4 DCTLNW
      PARAMETER (DCTLNW =58)
C       DcTypeMatrix
      INTEGER*4 DCTMX
      PARAMETER (DCTMX =59)
C       DcTypeMinBoundingExtension
      INTEGER*4 DCTMBE
      PARAMETER (DCTMBE =60)
C       DcTypeMarkerFont
      INTEGER*4 DCTMKF
      PARAMETER (DCTMKF =61)
C       DcTypeMarkerGlyph
      INTEGER*4 DCTMKG
      PARAMETER (DCTMKG =62)
C       DcTypeMarkerScale
      INTEGER*4 DCTMKS
      PARAMETER (DCTMKS =63)
C       DcTypeNameSet
      INTEGER*4 DCTNMS
      PARAMETER (DCTNMS =64)
C       DcTypeNURBSurface
      INTEGER*4 DCTNSF
      PARAMETER (DCTNSF =66)
C       DcTypeParallel
      INTEGER*4 DCTPAR
      PARAMETER (DCTPAR =69)
C       DcTypePatch
      INTEGER*4 DCTPAT
      PARAMETER (DCTPAT =70)
C       DcTypePickID
      INTEGER*4 DCTPKI
      PARAMETER (DCTPKI =71)
C       DcTypePickabilitySwitch
      INTEGER*4 DCTPKS
      PARAMETER (DCTPKS =72)
C       DcTypePerspective
      INTEGER*4 DCTPSP
      PARAMETER (DCTPSP =73)
C       DcTypePolygon
      INTEGER*4 DCTPGN
      PARAMETER (DCTPGN =74)
C       DcTypePolyline
      INTEGER*4 DCTPLN
      PARAMETER (DCTPLN =75)
C       DcTypePolymarker
```

```
      INTEGER*4 DCTPMK
      PARAMETER (DCTPMK =76)
C       DcTypePolygonMesh
      INTEGER*4 DCTPMH
      PARAMETER (DCTPMH =77)
C       DcTypePopAttributes
      INTEGER*4 DCTPPA
      PARAMETER (DCTPPA =78)
C       DcTypePrimitiveSurface
      INTEGER*4 DCTPMS
      PARAMETER (DCTPMS =79)
C       DcTypeProjection
      INTEGER*4 DCTPRJ
      PARAMETER (DCTPRJ =80)
C       DcTypePushAttributes
      INTEGER*4 DCTPHA
      PARAMETER (DCTPHA =81)
C       DcTypeReflectionSwitch
      INTEGER*4 DCTRFS
      PARAMETER (DCTRFS =82)
C       DcTypeRepType
      INTEGER*4 DCTRT
      PARAMETER (DCTRT =83)
C       DcTypeShadowSwitch
      INTEGER*4 DCTSHS
      PARAMETER (DCTSHS =84)
C       DcTypeShadeIndex
      INTEGER*4 DCTSHI
      PARAMETER (DCTSHI =85)
C       DcTypeSimplePolygon
      INTEGER*4 DCTSPN
      PARAMETER (DCTSPN =86)
C       DcTypeSimplePolygonMesh
      INTEGER*4 DCTSPM
      PARAMETER (DCTSPM =87)
C       DcTypeSpecularColor
      INTEGER*4 DCTSPC
      PARAMETER (DCTSPC =91)
C       DcTypeSpecularFactor
      INTEGER*4 DCTSPF
      PARAMETER (DCTSPF =92)
C       DcTypeSpecularIntens
      INTEGER*4 DCTSPI
      PARAMETER (DCTSPI =93)
C       DcTypeSpecularSwitch
      INTEGER*4 DCTSPS
      PARAMETER (DCTSPS =94)
C       DcTypeTorus
      INTEGER*4 DCTTOR
      PARAMETER (DCTTOR =98)
C       DcTypeTriangleMesh
      INTEGER*4 DCTTRM
      PARAMETER (DCTTRM =100)
C       DcTypeTranspColor
      INTEGER*4 DCTTPC
      PARAMETER (DCTTPC =101)
```

```
C       DcTypeTranspIntens
      INTEGER*4 DCTTPI
      PARAMETER (DCTTPI =102)
C       DcTypeTranspSwitch
      INTEGER*4 DCTTPS
      PARAMETER (DCTTPS =103)
C       DcTypeText
      INTEGER*4 DCTTXT
      PARAMETER (DCTTXT =104)
C       DcTypeTextAlign
      INTEGER*4 DCTTAL
      PARAMETER (DCTTAL =105)
C       DcTypeTextExpansionFactor
      INTEGER*4 DCTTEF
      PARAMETER (DCTTEF =106)
C       DcTypeTextFont
      INTEGER*4 DCTTFT
      PARAMETER (DCTTFT =107)
C       DcTypeTextHeight
      INTEGER*4 DCTTHT
      PARAMETER (DCTTHT =108)
C       DcTypeTextPath
      INTEGER*4 DCTTPH
      PARAMETER (DCTTPH =109)
C       DcTypeTextPrecision
      INTEGER*4 DCTTPN
      PARAMETER (DCTTPN =110)
C       DcTypeTextSpace
      INTEGER*4 DCTTSP
      PARAMETER (DCTTSP =111)
C       DcTypeTextUpVector
      INTEGER*4 DCTTUV
      PARAMETER (DCTTUV =112)
C       DcTypeAttAmbientSwitch
      INTEGER*4 DCTAAW
      PARAMETER (DCTAAW =115)
C       DcTypeAttDiffuseSwitch
      INTEGER*4 DCTADS
      PARAMETER (DCTADS =116)
C       DcTypeAttSpecularSwitch
      INTEGER*4 DCTASW
      PARAMETER (DCTASW =117)
C       DcTypeAttDiffuseColor
      INTEGER*4 DCTADC
      PARAMETER (DCTADC =118)
C       DcTypeAttSpecularColor
      INTEGER*4 DCTASC
      PARAMETER (DCTASC =119)
C       DcTypeAttDiffuseIntens
      INTEGER*4 DCTADI
      PARAMETER (DCTADI =120)
C       DcTypeAttSpecularIntens
      INTEGER*4 DCTASI
      PARAMETER (DCTASI =121)
C       DcTypeAttSpecularFactor
      INTEGER*4 DCTASF
```

```
      PARAMETER (DCTASF =122)
C       DcTypeAttRepType
      INTEGER*4 DCTART
      PARAMETER (DCTART =123)
C       DcTypeAttInterpType
      INTEGER*4 DCTAIT
      PARAMETER (DCTAIT =124)
C       DcTypeAttLcstowcsmat
      INTEGER*4 DCTALM
      PARAMETER (DCTALM =125)
C       DcTypeAttBackfaceCullSwitch
      INTEGER*4 DCTABS
      PARAMETER (DCTABS =126)
C       DcTypeBackfaceCullable
      INTEGER*4 DCTBFC
      PARAMETER (DCTBFC =127)
C       DcTypeAttBackfaceCullable
      INTEGER*4 DCTABC
      PARAMETER (DCTABC =128)
C       DcTypeAttLightIntens
      INTEGER*4 DCTALI
      PARAMETER (DCTALI =129)
C       DcTypeAttLightColor
      INTEGER*4 DCTALC
      PARAMETER (DCTALC =131)
C       DcTypeAttShadowSwitch
      INTEGER*4 DCTASH
      PARAMETER (DCTASH =132)
C       DcTypeAttReflectionSwitch
      INTEGER*4 DCTARW
      PARAMETER (DCTARW =133)
C       DcTypeAttGlbRndMaxObjs
      INTEGER*4 DCTAMO
      PARAMETER (DCTAMO =134)
C       DcTypeAttGlbRndMaxSub
      INTEGER*4 DCTARS
      PARAMETER (DCTARS =135)
C       DcTypeAttGlbRndRayLevel
      INTEGER*4 DCTARL
      PARAMETER (DCTARL =136)
C       DcTypeAttTranspColor
      INTEGER*4 DCTATC
      PARAMETER (DCTATC =137)
C       DcTypeAttTranspIntens
      INTEGER*4 DCTATI
      PARAMETER (DCTATI =138)
C       DcTypeAttTranspSwitch
      INTEGER*4 DCTATS
      PARAMETER (DCTATS =139)
C       DcTypeAttTextAlignment
      INTEGER*4 DCTATA
      PARAMETER (DCTATA =145)
C       DcTypeAttTextExpansion
      INTEGER*4 DCTATE
      PARAMETER (DCTATE =146)
C       DcTypeAttTextFont
```

```
      INTEGER*4 DCTATF
      PARAMETER (DCTATF =147)
C       DcTypeAttTextHeight
      INTEGER*4 DCTATH
      PARAMETER (DCTATH =148)
C       DcTypeAttTextPath
      INTEGER*4 DCTATP
      PARAMETER (DCTATP =149)
C       DcTypeAttTextSpace
      INTEGER*4 DCTAXS
      PARAMETER (DCTAXS =150)
C       DcTypeAttTextUpVector
      INTEGER*4 DCTATU
      PARAMETER (DCTATU =151)
C       DcTypeAttTextPrecision
      INTEGER*4 DCTATR
      PARAMETER (DCTATR =152)
C       DcTypeRayinttri
      INTEGER*4 DCTRIT
      PARAMETER (DCTRIT =153)
C       DcTypeRayintmshtri
      INTEGER*4 DCTRMT
      PARAMETER (DCTRMT =154)
C       DcTypeNull
      INTEGER*4 DCTNUL
      PARAMETER (DCTNUL =155)
C       DcTypeAttLineType
      INTEGER*4 DCTALT
      PARAMETER (DCTALT =156)
C       DcTypeAttLineWidth
      INTEGER*4 DCTALW
      PARAMETER (DCTALW =157)
C       DcTypeAttMarkerScale
      INTEGER*4 DCTAMS
      PARAMETER (DCTAMS =158)
C       DcTypeAttMarkerGlyph
      INTEGER*4 DCTAMG
      PARAMETER (DCTAMG =159)
C       DcTypeAttMarkerFont
      INTEGER*4 DCTAMF
      PARAMETER (DCTAMF =160)
C       DcTypeAttExecSet
      INTEGER*4 DCTAES
      PARAMETER (DCTAES =161)
C       DcTypeAttNameSet
      INTEGER*4 DCTANS
      PARAMETER (DCTANS =162)
C       DcTypeAttFilter
      INTEGER*4 DCTAFT
      PARAMETER (DCTAFT =163)
C       DcTypeAttInvisibilitySwitch
      INTEGER*4 DCTAIS
      PARAMETER (DCTAIS =164)
C       DcTypeAttPickabilitySwitch
      INTEGER*4 DCTAPS
      PARAMETER (DCTAPS =165)
```

```
C       DcTypeAttClipSwitch
      INTEGER*4 DCTACS
      PARAMETER (DCTACS =168)
C       DcTypeAttClipVolume
      INTEGER*4 DCTACV
      PARAMETER (DCTACV =169)
C       DcTypeBoundingVolumeSwitch
      INTEGER*4 DCTBVS
      PARAMETER (DCTBVS =170)
C       DcTypeAttBoundingVolumeSwitch
      INTEGER*4 DCTABV
      PARAMETER (DCTABV =171)
C       DcTypeAttMinBoundingExtension
      INTEGER*4 DCTAME
      PARAMETER (DCTAME =172)
C       DcTypeAttPickID
      INTEGER*4 DCTAPI
      PARAMETER (DCTAPI =173)
C       DcTypeTriangleList
      INTEGER*4 DCTTRL
      PARAMETER (DCTTRL =175)
C       DcTypeAttDepthCue
      INTEGER*4 DCTADU
      PARAMETER (DCTADU =176)
C       DcTypeAttDepthCueSwitch
      INTEGER*4 DCTADW
      PARAMETER (DCTADW =177)
C       DcTypeAttHiddenSurfaceSwitch
      INTEGER*4 DCTAHW
      PARAMETER (DCTAHW =178)
C       DcTypeSubDivSpec
      INTEGER*4 DCTSDS
      PARAMETER (DCTSDS =179)
C       DcTypeAttSubDivSpec
      INTEGER*4 DCTASD
      PARAMETER (DCTASD =180)
C       DcTypeAttLightType
      INTEGER*4 DCTALY
      PARAMETER (DCTALY =182)
C       DcTypeSurfaceShader
      INTEGER*4 DCTSSH
      PARAMETER (DCTSSH =183)
C       DcTypeAttSurfaceShader
      INTEGER*4 DCTASS
      PARAMETER (DCTASS =184)
C       DcTypeLineList
      INTEGER*4 DCTLNL
      PARAMETER (DCTLNL =185)
C       DcTypePointList
      INTEGER*4 DCTPTL
      PARAMETER (DCTPTL =186)
C       DcTypeAttShadeIndex
      INTEGER*4 DCTASN
      PARAMETER (DCTASN =187)
C       DcTypeSphereList
      INTEGER*4 DCTSPH
```

```
      PARAMETER (DCTSPH =189)
C       DcTypeCylinderList
      INTEGER*4 DCTCYL
      PARAMETER (DCTCYL =190)
C       DcTypeStereoSwitch
      INTEGER*4 DCTSTS
      PARAMETER (DCTSTS =193)
C       DcTypeAttStereoSwitch
      INTEGER*4 DCTAOS
      PARAMETER (DCTAOS =194)
C       DcTypeStereo
      INTEGER*4 DCTSTR
      PARAMETER (DCTSTR =195)
C       DcTypeAttStereo
      INTEGER*4 DCTASR
      PARAMETER (DCTASR = 196)
C       DcTypeAmbientIntens
      INTEGER*4 DCTAMI
      PARAMETER (DCTAMI =199)
C       DcTypeAttAmbientIntens
      INTEGER*4 DCTAAI
      PARAMETER (DCTAAI =200)
C       DcTypeVarTriangleMesh
      INTEGER*4 DCTVTM
      PARAMETER (DCTVTM =201)
C       DcTypeVarLineList
      INTEGER*4 DCTVLL
      PARAMETER (DCTVLL =202)
C       DcTypeVarPointList
      INTEGER*4 DCTVPL
      PARAMETER (DCTVPL =203)
C       DcTypeVarSimplePolygonMesh
      INTEGER*4 DCTVSM
      PARAMETER (DCTVSM =204)
```

Filename: DORE

```
C************************************************************************
CCopyright (C) 1989, by Stardent Computer Corp.
C        All Rights Reserved
CThis program is a  trade secret of  Stardent Computer Corp. and it is not
Cto be reproduced, published, disclosed  to others, copied, adapted,
Cdistributed, or displayed without the prior authorization of Stardent
CComputer Corp.  Licensee agrees  to  attach  or  embed  this  Notice on
Call copies  of the program, including partial copies or modified versions
Cthereof.
C************************************************************************
C
C**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**
C**                                                                      **
C**  This file contains all the Dore FORTRAN constant declarations.      **
C**                                                                      **
C************************************************************************

      INTEGER*4 DCNULL
      COMMON /DCNULL/DCNULL

C DtUpdateType
C       DcUpdateAll:
      INTEGER*4 DCUALL
      PARAMETER (DCUALL =0)
C       DcUpdateDisplay
      INTEGER*4 DCUDIS
      PARAMETER (DCUDIS =1)

C DtFlag
C       DcFalse
      INTEGER*4 DCFALS
      PARAMETER (DCFALS =0)
C       DcTrue
      INTEGER*4 DCTRUE
      PARAMETER (DCTRUE =1)

C DtSwitch
C       DcOff
      INTEGER*4 DCOFF
      PARAMETER (DCOFF =0)
C       DcOn
      INTEGER*4 DCON
      PARAMETER (DCON =1)

C DtSurface
C       DcSphere
      INTEGER*4 DCSPHR
      PARAMETER (DCSPHR =0)
C       DcCylinder
```

```
      INTEGER*4 DCCYL
      PARAMETER (DCCYL =1)
C       DcBox
      INTEGER*4 DCBOX
      PARAMETER (DCBOX =2)
C       DcCone
      INTEGER*4 DCCONE
      PARAMETER (DCCONE =3)

C DtRepType
C       DcPoints
      INTEGER*4 DCPNTS
      PARAMETER (DCPNTS =0)
C       DcWireframe
      INTEGER*4 DCWIRE
      PARAMETER (DCWIRE =1)
C       DcSurface
      INTEGER*4 DCSURF
      PARAMETER (DCSURF =2)
C       DcOutlines
      INTEGER*4 DCOUTL
      PARAMETER (DCOUTL =3)

C DtInterpType
C       DcConstantShade
      INTEGER*4 DCCNSH
      PARAMETER (DCCNSH =0)
C       DcVertexShade
      INTEGER*4 DCVXSH
      PARAMETER (DCVXSH =1)
C       DcSurfaceShade
      INTEGER*4 DCSFSH
      PARAMETER (DCSFSH =2)

C DtAxis
C       DcXAxis
      INTEGER*4 DCXAX
      PARAMETER (DCXAX =0)
C       DcYAxis
      INTEGER*4 DCYAX
      PARAMETER (DCYAX =1)
C       DcZAxis
      INTEGER*4 DCZAX
      PARAMETER (DCZAX =2)

C DtMajorPlane
C       DcXY
      INTEGER*4 DCXY
      PARAMETER (DCXY =0)
C       DcYZ
      INTEGER*4 DCYZ
      PARAMETER (DCYZ =1)
C       DcXZ
      INTEGER*4 DCXZ
      PARAMETER (DCXZ =2)
```

```
C DtColorModel
C       DcRGB
      INTEGER*4 DCRGB
      PARAMETER (DCRGB =0)

C DtVisualType
C       DcStaticGrey
      INTEGER*4 DCSTCG
      PARAMETER (DCSTCG =0)
C       DcGreyScale
      INTEGER*4 DCGRYS
      PARAMETER (DCGRYS =1)
C       DcStaticColor
      INTEGER*4 DCSTCC
      PARAMETER (DCSTCC =2)
C       DcPseudoColor
      INTEGER*4 DCPSUC
      PARAMETER (DCPSUC =3)
C       DcTrueColor
      INTEGER*4 DCTRUC
      PARAMETER (DCTRUC =4)
C       DcDirectColor
      INTEGER*4 DCDRCC
      PARAMETER (DCDRCC =5)

C DtShadeMode
C       DcRange
      INTEGER*4 DCRNG
      PARAMETER (DCRNG =0)
C       DcComponent
      INTEGER*4 DCCOMP
      PARAMETER (DCCOMP =1)

C DtVertexType
C       DcLoc
      INTEGER*4 DCL
      PARAMETER (DCL =0)
C       DcLocNorm
      INTEGER*4 DCLN
      PARAMETER (DCLN =1)
C       DcLocClr
      INTEGER*4 DCLC
      PARAMETER (DCLC =2)
C       DcLocNrmClr
      INTEGER*4 DCLNC
      PARAMETER (DCLNC =3)

C DtCtrlPointType
C       DcCtr
      INTEGER*4 DCCTR
      PARAMETER (DCCTR =0)
C       DcCtrClr
      INTEGER*4 DCCTRC
      PARAMETER (DCCTRC =1)

C DtCompType
```

```
C       DcPreConcatenate
      INTEGER*4 DCPREC
      PARAMETER (DCPREC =0)
C       DcPostConcatenate
      INTEGER*4 DCPSTC
      PARAMETER (DCPSTC =1)
C       DcReplace
      INTEGER*4 DCREPL
      PARAMETER (DCREPL =2)

C DtRenderStyle;
C       DcRealTime
      INTEGER*4 DCRLTM
      PARAMETER (DCRLTM =0)
C       DcProductionTime
      INTEGER*4 DCPRTM
      PARAMETER (DCPRTM =2)

C DtShapeType;
C       DcConvex
      INTEGER*4 DCCNVX
      PARAMETER (DCCNVX =0)
C       DcConcave
      INTEGER*4 DCCNCV
      PARAMETER (DCCNCV =1)
C       DcComplex
      INTEGER*4 DCCPLX
      PARAMETER (DCCPLX =2)

C DtRelPosition;
C       DcBeginning
      INTEGER*4 DCBEG
      PARAMETER (DCBEG =0)
C       DcEnd
      INTEGER*4 DCEND
      PARAMETER (DCEND =1)
C       DcCurrent
      INTEGER*4 DCCUR
      PARAMETER (DCCUR =2)

C DtProjectionType;
C       DcParallel
      INTEGER*4 DCPRLL
      PARAMETER (DCPRLL =0)
C       DcPerspective
      INTEGER*4 DCPRSP
      PARAMETER (DCPRSP =1)

C DtClipOperator;
C       DcClipAll
      INTEGER*4 DCCALL
      PARAMETER (DCCALL =0)
C       DcClipAnd
      INTEGER*4 DCCAND
      PARAMETER (DCCAND =1)
C       DcClipAndReverse
```

```
      INTEGER*4 DCCARV
      PARAMETER (DCCARV =2)
C       DcClipNoOp
      INTEGER*4 DCCNOP
      PARAMETER (DCCNOP =3)
C       DcClipAndInverted
      INTEGER*4 DCCAIN
      PARAMETER (DCCAIN =4)
C       DcClipReplace
      INTEGER*4 DCCREP
      PARAMETER (DCCREP =5)
C       DcClipXor
      INTEGER*4 DCCXOR
      PARAMETER (DCCXOR =6)
C       DcClipOr
      INTEGER*4 DCCOR
      PARAMETER (DCCOR =7)
C       DcClipNor
      INTEGER*4 DCCNOR
      PARAMETER (DCCNOR =8)
C       DcClipEqv
      INTEGER*4 DCCEQV
      PARAMETER (DCCEQV =9)
C       DcClipInvertVolume
      INTEGER*4 DCCIVV
      PARAMETER (DCCIVV =10)
C       DcClipOrReverse
      INTEGER*4 DCCORR
      PARAMETER (DCCORR =11)
C       DcClipInvert
      INTEGER*4 DCCINV
      PARAMETER (DCCINV =12)
C       DcClipOrInverted
      INTEGER*4 DCCORI
      PARAMETER (DCCORI =13)
C       DcClipNAnd
      INTEGER*4 DCCNAN
      PARAMETER (DCCNAN =14)
C       DcClipNone
      INTEGER*4 DCCNON
      PARAMETER (DCCNON =15)
C       DcClip

C DtNameType;
C       DcNameNone
      INTEGER*4 DCNNON
      PARAMETER (DCNNON =0)
C       DcNameInteger
      INTEGER*4 DCNINT
      PARAMETER (DCNINT =1)
C       DcNameString
      INTEGER*4 DCNSTR
      PARAMETER (DCNSTR =2)

C DtPickPathOrder;
C       DcTopFirst
```

```
      INTEGER*4 DCTOPF
      PARAMETER (DCTOPF =0)
C       DcBottomFirst
      INTEGER*4 DCBOTF
      PARAMETER (DCBOTF =1)

C DtHitStatus;
C       DcHitAccept
      INTEGER*4 DCHACC
      PARAMETER (DCHACC =0)
C       DcHitReject
      INTEGER*4 DCHREJ
      PARAMETER (DCHREJ =1)
C       DcHitOverwrite
      INTEGER*4 DCHOVW
      PARAMETER (DCHOVW =2)

C DtFilter;
C       InvisibilityInclusion
      INTEGER*4 DCINVI
      PARAMETER (DCINVI =0)
C       InvisibilityExclusion
      INTEGER*4 DCINVE
      PARAMETER (DCINVE =1)
C       DcPickabilityInclusion
      INTEGER*4 DCPCKI
      PARAMETER (DCPCKI =2)
C       DcPickabilityExclusion
      INTEGER*4 DCPCKE
      PARAMETER (DCPCKE =3)

C DtSetOperation;
C       DcSetAdd
      INTEGER*4 DCSADD
      PARAMETER (DCSADD =0)
C       DcSetDelete
      INTEGER*4 DCSDEL
      PARAMETER (DCSDEL =1)
C       DcSetInvert
      INTEGER*4 DCSINV
      PARAMETER (DCSINV =2)
C       DcSetReplace
      INTEGER*4 DCSREP
      PARAMETER (DCSREP =3)

C DtLineType;
C       DcLineTypeSolid
      INTEGER*4 DCLTS
      PARAMETER (DCLTS =0)
C       DcLineTypeDash
      INTEGER*4 DCLTDS
      PARAMETER (DCLTDS =1)
C       DcLineTypeDot
      INTEGER*4 DCLTDT
      PARAMETER (DCLTDT =2)
C       DcLineTypeDotDash
```

```
      INTEGER*4 DCLTDD
      PARAMETER (DCLTDD =3)

C DtGroupNetworkStatus;
C       DcGroupOk
      INTEGER*4 DCGOK
      PARAMETER (DCGOK =0)
C       DcGroupBad
      INTEGER*4 DCGBAD
      PARAMETER (DCGBAD =1)
C       DcGroupCond
      INTEGER*4 DCGCND
      PARAMETER (DCGCND =2)

C DtErrorStatus;
C       DcErrorMinor
      INTEGER*4 DCEMNR
      PARAMETER (DCEMNR =0)
C       DcErrorSevere
      INTEGER*4 DCESEV
      PARAMETER (DCESEV =1)
C       DcErrorFatal
      INTEGER*4 DCEFAT
      PARAMETER (DCEFAT =2)

C DtReadStatus;
C       DcReadOk
      INTEGER*4 DCROK
      PARAMETER (DCROK =0)
C       DcReadTrunc
      INTEGER*4 DCRTRN
      PARAMETER (DCRTRN =1)
C       DcReadUnsuc
      INTEGER*4 DCRFAL
      PARAMETER (DCRFAL =2)

C DtObjectStatus;
C       DcObjectValid
      INTEGER*4 DCOVLD
      PARAMETER (DCOVLD =0)
C       DcObjectInvalid
      INTEGER*4 DCOINV
      PARAMETER (DCOINV =1)
C       DcObjectDeleted
      INTEGER*4 DCODEL
      PARAMETER (DCODEL =2)

C subdivision types
C       DcSubDivFixed
      INTEGER*4 DCSDFX
      PARAMETER (DCSDFX =0)
C       DcSubDivAbsolute
      INTEGER*4 DCSDAB
      PARAMETER (DCSDAB =1)
C       DcSubDivRelative
      INTEGER*4 DCSDRL
```

```
      PARAMETER (DCSDRL =2)
C       DcSubDivSegments
      INTEGER*4 DCSDSG
      PARAMETER (DCSDSG =3)

C pick status
C       DcPickBadStatus
      INTEGER*4 DCPBAD
      PARAMETER (DCPBAD =1)
C       DcPickListOverflow
      INTEGER*4 DCPLOV
      PARAMETER (DCPLOV =2)
C       DcPickIndexOverflow
      INTEGER*4 DCPIOV
      PARAMETER (DCPIOV =4)

C marker types
C       DcMarkerPoint
      INTEGER*4 DCMKPT
      PARAMETER (DCMKPT =-1)
C       DcMarkerPlus
      INTEGER*4 DCMKPL
      PARAMETER (DCMKPL =1)
C       DcMarkerStar
      INTEGER*4 DCMKST
      PARAMETER (DCMKST =2)
C       DcMarkerO
      INTEGER*4 DCMKO
      PARAMETER (DCMKO =3)
C       DcMarkerX
      INTEGER*4 DCMKX
      PARAMETER (DCMKX =4)
C       DcMarkerDiamond
      INTEGER*4 DCMKDD
      PARAMETER (DCMKDD =5)
C       DcMarkerSquare
      INTEGER*4 DCMKSQ
      PARAMETER (DCMKSQ =6)
C       DcMarkerTriangle
      INTEGER*4 DCMKTR
      PARAMETER (DCMKTR =7)

C DtRasterType
C       DcRasterRGB
      INTEGER*4 DCRRGB
      PARAMETER (DCRRGB =0)
C       DcRasterRGBA
      INTEGER*4 DCRRA
      PARAMETER (DCRRA  =1)
C       DcRasterRGBAZ
      INTEGER*4 DCRRAZ
      PARAMETER (DCRRAZ =2)
C       DcRasterRGBZ
      INTEGER*4 DCRRZ
      PARAMETER (DCRRZ  =3)
C       DcRasterA
```

```
      INTEGER*4 DCRA
      PARAMETER (DCRA   =4)
C       DcRasterZ
      INTEGER*4 DCRZ
      PARAMETER (DCRZ   =5)
C       DcRasterSpecial
      INTEGER*4 DCRSPC
      PARAMETER (DCRSPC =99)

C DtTextureAntiAliasMode:
C       DcTextureAntiAliasNone
      INTEGER*4 DCTAAN
      PARAMETER (DCTAAN =0)
C       DcTextureAntiAliasMIP
      INTEGER*4 DCTAAM
      PARAMETER (DCTAAM =1)
C       DcTextureAntiAliasAdaptive
      INTEGER*4 DCTAAA
      PARAMETER (DCTAAA =2)

C DtExtendMode
C       DcExtendNone
      INTEGER*4 DCXNON
      PARAMETER (DCXNON =0)
C       DcExtendBlack
      INTEGER*4 DCXBLK
      PARAMETER (DCXBLK =1)
C       DcExtendClamp
      INTEGER*4 DCXCLP
      PARAMETER (DCXCLP =2)
C       DcExtendRepeat
      INTEGER*4 DCXRP
      PARAMETER (DCXRP =3)

C DtMapOperator
C       DcMapReplace
      INTEGER*4 DCMR
      PARAMETER (DCMR =0)
C       DcMapAdd
      INTEGER*4 DCMADD
      PARAMETER (DCMADD =1)

C DtTextureOperator
C       DcTextureReplace
      INTEGER*4 DCTREP
      PARAMETER (DCTREP =0)
C       DcTextureMultiply
      INTEGER*4 DCTMUL
      PARAMETER (DCTMUL =1)
C       DcTextureBlend
      INTEGER*4 DCTBLD
      PARAMETER (DCTBLD =2)
C       DcTextureAdd
      INTEGER*4 DCTADD
      PARAMETER (DCTADD =3)
```

```
C Standard matrices for cubic curves and patches
C       DcBezier4, DcHermite4, DcBSpline4
      INTEGER*4 DCMXB4, DCMXH4, DCMXBS
      COMMON /DCMXS/DCMXB4, DCMXH4, DCMXBS

C Standard light type objects
C       DcLightAmbient, DcLightInfinite,
C       DcLightPoint, DcLightPointAttn,
C       DcLightSpot, DcLightSpotAttn
      INTEGER*4 DCLTAM, DCLTIN, DCLTPT, DCLTPA, DCLTSP, DCLTSA
      COMMON /DCLTPS/DCLTAM, DCLTIN, DCLTPT, DCLTPA, DCLTSP, DCLTSA

C Standard surface shader objects
C       DcShaderConstant, DcShaderLightSource
      INTEGER*4 DCSHCN, DCSHLS
      COMMON /DCSHS/DCSHCN, DCSHLS

C Standard pick call-backs
C       DcPickFirst, DcPickAll, DcPickClosest
      INTEGER*4 DCPKFR, DCPKAL, DCPKCL
      COMMON /DCPKS/DCPKFR, DCPKAL, DCPKCL

C Standard input slots
C       DcTransXSlot, YSlot, ZSlot
      INTEGER*4 DCISTX, DCISTY, DCISTZ
C       DcScaleXSlot, YSlot, ZSlot
      INTEGER*4 DCISSX, DCISSY, DCISSZ
C       DcRotXSlot, YSlot, ZSlot
      INTEGER*4 DCISRX, DCISRY, DCISRZ
C       DcUndoSlot, DcUpdateSlot
      INTEGER*4 DCISUD, DCISUP
      COMMON /DCISS/DCISTX, DCISTY, DCISTZ, DCISSX, DCISSY, DCISSZ,
     + DCISRX, DCISRY, DCISRZ, DCISUD, DCISUP

C Super Sample Filters
C       DcFilterBox
      INTEGER*4 DCFBOX
      COMMON /DCSFLT/ DCFBOX

C Standard Texture Mapping Functions
C       DcStdBumpMap, DcStdTableLookup, DcStd3dTableLookup,
C       DcStdSphereEnvironMap, DcStdCubeEnvironMap
      INTEGER*4 DCSBM
      INTEGER*4 DCSTLU
      INTEGER*4 DCS3TL
      INTEGER*4 DCSSEM
      INTEGER*4 DCSCEM
      COMMON /DCSMAP/ DCSBM, DCSTLU, DCS3TL, DCSSEM, DCSCEM

C Standard Raster Callback Delete Data Functions
C       DcDeleteData
      INTEGER*4 DCDELD
      COMMON /DCDELC/ DCDELD

C Routine declarations
```

```
C       DdPickObjs              returns void
      EXTERNAL  DDPO
C       DdPick                  returns void  -- OBSELETE
      EXTERNAL  DDP
C       DdInqColorEntries       returns void
      EXTERNAL  DDQCE
C       DdInqColorTableSize     returns DtInt
      EXTERNAL  DDQCTS
      INTEGER*4 DDQCTS
C       DdInqExtent           returns void
      EXTERNAL  DDQE
C       DdInqFrame            returns DtObject
      EXTERNAL  DDQFR
      INTEGER*4 DDQFR
C       DdInqFonts            returns void
      EXTERNAL  DDQFT
C       DdInqNumFonts         returns DtInt
      EXTERNAL  DDQNF
      INTEGER*4 DDQNF
C       DdInqPickAperture     returns void
      EXTERNAL  DDQPA
C       DdInqPickCallback     returns DtObject
      EXTERNAL  DDQPC
      INTEGER*4 DDQPC
C       DdInqPickPathOrder    returns DtPickPathOrder
      EXTERNAL  DDQPPO
      INTEGER*4 DDQPPO
C       DcInqPixelData          returns DtFlag
      EXTERNAL   DDQPXD
      INTEGER*4 DDQPXD
C       DdInqResolution       returns void
      EXTERNAL  DDQR
C       DdInqShadeMode        returns DtShadeMode
      EXTERNAL  DDQSM
      INTEGER*4 DDQSM
C       DdInqShadeRanges      returns void
      EXTERNAL  DDQSR
C       DdInqViewport         returns void
      EXTERNAL  DDQV
C       DdInqVisualType       returns DtVisualType
      EXTERNAL  DDQVT
      INTEGER*4 DDQVT
C       DdSetColorEntries     returns void
      EXTERNAL  DDSCE
C       DdSetViewPort         returns void
      EXTERNAL  DDSDV
C       DdSetFrame            returns void
      EXTERNAL  DDSF
C       DdSetPickAperture     returns void
      EXTERNAL  DDSPA
C       DdSetPickCallback     returns void
      EXTERNAL  DDSPCB
C       DdSetPickPathOrder    returns void
      EXTERNAL  DDSPPO
C       DdSetShadeMode        returns void
      EXTERNAL  DDSSM
```

```
C       DdSetShadeRanges      returns void
      EXTERNAL  DDSSR
C       DdUpdate              returns void
      EXTERNAL  DDU
C       DfInqBoundary         returns void
      EXTERNAL  DFQB
C       DfInqJust             returns void
      EXTERNAL  DFQJ
C       DfInqViewGroup        returns DtObject
      EXTERNAL  DFQVG
      INTEGER*4 DFQVG
C       DfSetBoundary         returns void
      EXTERNAL  DFSB
C       DfSetJust             returns void
      EXTERNAL  DFSJ
C       DfUpdate              returns void
      EXTERNAL  DFU
C       DgAddObj              returns void
      EXTERNAL  DGAO
C       DgAddObjToGroup       returns void
      EXTERNAL  DGAOG
C       DgCheck               returns DtGroupNetworkStatus
      EXTERNAL  DGCK
      INTEGER*4 DGCK
C       DgClose               returns DtObject
      EXTERNAL  DGCS
      INTEGER*4 DGCS
C       DgDelEle              returns void
      EXTERNAL  DGDE
C       DgDelEleBetweenLabels returns DtFlag
      EXTERNAL  DGDEL
      INTEGER*4 DGDEL
C       DgDelEleRange         returns void
      EXTERNAL  DGDER
C       DgEmpty               returns void
      EXTERNAL  DGE
C       DgOpen                returns void
      EXTERNAL  DGO
C       DgInqElePtr           returns DtInt
      EXTERNAL  DGQEP
      INTEGER*4 DGQEP
C       DgInqOpen             returns DtObject
      EXTERNAL  DGQO
      INTEGER*4 DGQO
C       DgInqObjAtPos         returns DtObject
      EXTERNAL  DGQOP
      INTEGER*4 DGQOP
C       DgInqSize             returns DtInt
      EXTERNAL  DGQS
      INTEGER*4 DGQS
C       DgReplaceObj          returns void
      EXTERNAL  DGRO
C       DgReplaceObjInGroup   returns void
      EXTERNAL  DGROG
C       DgSetElePtr           returns void
      EXTERNAL  DGSEP
```

```
C       DgSetElePtrRelLabel   returns DtFlag
      EXTERNAL  DGSEPL
      INTEGER*4 DGSEPL
C       DoAmbientIntens       returns DtObject
      EXTERNAL  DOAMBI
      INTEGER*4 DOAMBI
C       DoAmbientSwitch       returns DtObject
      EXTERNAL  DOAMBS
      INTEGER*4 DOAMBS
C       DoAnnoText            returns DtObject
      EXTERNAL  DOANNT
      INTEGER*4 DOANNT
C       DoBackfaceCullable    returns DtObject
      EXTERNAL  DOBFC
      INTEGER*4 DOBFC
C       DoBackfaceCullSwitch  returns DtObject
      EXTERNAL  DOBFCS
      INTEGER*4 DOBFCS
C       DoBoundingVol         returns DtObject
      EXTERNAL  DOBV
      INTEGER*4 DOBV
C       DoBoundingVolSwitch   returns DtObject
      EXTERNAL  DOBVS
      INTEGER*4 DOBVS
C       DoCallback            returns DtObject
      EXTERNAL  DOCB
      INTEGER*4 DOCB
C       DoCamera              returns DtObject
      EXTERNAL  DOCM
      INTEGER*4 DOCM
C       DoCameraMatrix        returns DtObject
      EXTERNAL  DOCMX
      INTEGER*4 DOCMX
C       DoClipSwitch          returns DtObject
      EXTERNAL  DOCS
      INTEGER*4 DOCS
C       DoClipVol             returns DtObject
      EXTERNAL  DOCV
      INTEGER*4 DOCV
C       DoDevice              returns DtObject
      EXTERNAL  DOD
      INTEGER*4 DOD
C       DoDepthCue            returns DtObject
      EXTERNAL  DODC
      INTEGER*4 DODC
C       DoDepthCueSwitch      returns DtObject
      EXTERNAL  DODCS
      INTEGER*4 DODCS
C       DoDiffuseColor        returns DtObject
      EXTERNAL  DODIFC
      INTEGER*4 DODIFC
C       DoDiffuseIntens       returns DtObject
      EXTERNAL  DODIFI
      INTEGER*4 DODIFI
C       DoDiffuseSwitch       returns DtObject
      EXTERNAL  DODIFS
```

```
      INTEGER*4 DODIFS
C       DoDataPtr              returns DtObject
      EXTERNAL  DODP
      INTEGER*4 DODP
C       DoDataVal              returns DtObject
      EXTERNAL  DODV
      INTEGER*4 DODV
C       DoExecSet              returns DtObject
      EXTERNAL  DOES
      INTEGER*4 DOES
C       DoFileRaster           returns DtObject
      EXTERNAL  DOFRS
      INTEGER*4 DOFRS
C       DoFilter               returns DtObject
      EXTERNAL  DOFL
      INTEGER*4 DOFL
C       DoFrame                returns DtObject
      EXTERNAL  DOFR
      INTEGER*4 DOFR
C       DoGroup                returns DtObject
      EXTERNAL  DOG
      INTEGER*4 DOG
C       DoGlbRendMaxObjs       returns DtObject
      EXTERNAL  DOGRMO
      INTEGER*4 DOGRMO
C       DoGlbRendMaxSub        returns DtObject
      EXTERNAL  DOGRMS
      INTEGER*4 DOGRMS
C       DoGlbRendRayLevel      returns DtObject
      EXTERNAL  DOGRRL
      INTEGER*4 DOGRRL
C       DoGenerateTextureUV    returns DtObject
      EXTERNAL  DOGTUV
      INTEGER*4 DOGTUV
C       DoHiddenSurfSwitch     returns DtObject
      EXTERNAL  DOHSS
      INTEGER*4 DOHSS
C       DoInLineGroup          returns DtObject
      EXTERNAL  DOILG
      INTEGER*4 DOILG
C       DoInvisSwitch          returns DtObject
      EXTERNAL  DOINVS
      INTEGER*4 DOINVS
C       DoInputSlot            returns DtObject
      EXTERNAL  DOIS
      INTEGER*4 DOIS
C       DoInterpType           returns DtObject
      EXTERNAL  DOIT
      INTEGER*4 DOIT
C       DoLookAtFrom           returns DtObject
      EXTERNAL  DOLAF
      INTEGER*4 DOLAF
C       DoLightColor           returns DtObject
      EXTERNAL  DOLC
      INTEGER*4 DOLC
C       DoLightIntens          returns DtObject
```

```
      EXTERNAL  DOLI
      INTEGER*4 DOLI
C       DoLineList            returns DtObject
      EXTERNAL  DOLINL
      INTEGER*4 DOLINL
C       DoLabel               returns DtObject
      EXTERNAL  DOLL
      INTEGER*4 DOLL
C       DoLineType            returns DtObject
      EXTERNAL  DOLNT
      INTEGER*4 DOLNT
C       DoLight               returns DtObject
      EXTERNAL  DOLT
      INTEGER*4 DOLT
C       DoLightType           returns DtObject
      EXTERNAL  DOLTT
      INTEGER*4 DOLTT
C       DoLightAttenuation    returns DtObject
      EXTERNAL  DOLTA
      INTEGER*4 DOLTA
C       DoLightSpreadAngles   returns DtObject
      EXTERNAL  DOLTSA
      INTEGER*4 DOLTSA
C       DoLightSpreadExp      returns DtObject
      EXTERNAL  DOLTSE
      INTEGER*4 DOLTSE
C       DoLightSwitch         returns DtObject
      EXTERNAL  DOLTS
      INTEGER*4 DOLTS
C       DoLineWidth           returns DtObject
      EXTERNAL  DOLW
      INTEGER*4 DOLW
C       DoMatrix              returns DtObject
      EXTERNAL  DOM
      INTEGER*4 DOM
C       DoMinBoundingVolExt   returns DtObject
      EXTERNAL  DOMBVE
      INTEGER*4 DOMBVE
C       DoMarkerFont          returns DtObject
      EXTERNAL  DOMF
      INTEGER*4 DOMF
C       DoMarkerGlyph         returns DtObject
      EXTERNAL  DOMG
      INTEGER*4 DOMG
C       DoMarkerScale         returns DtObject
      EXTERNAL  DOMS
      INTEGER*4 DOMS
C       DoNURBSurf            returns DtObject
      EXTERNAL  DONRBS
      INTEGER*4 DONRBS
C       DoNameSet             returns DtObject
      EXTERNAL  DONS
      INTEGER*4 DONS
C       DoParallel            returns DtObject
      EXTERNAL  DOPAR
      INTEGER*4 DOPAR
```

```
C       DoPatch                 returns DtObject
      EXTERNAL  DOPAT
      INTEGER*4 DOPAT
C       DoPerspective           returns DtObject
      EXTERNAL  DOPER
      INTEGER*4 DOPER
C       DoPolygon               returns DtObject
      EXTERNAL  DOPGN
      INTEGER*4 DOPGN
C       DoPolygonMesh           returns DtObject
      EXTERNAL  DOPGNM
      INTEGER*4 DOPGNM
C       DoPickID                returns DtObject
      EXTERNAL  DOPID
      INTEGER*4 DOPID
C       DoPolyline              returns DtObject
      EXTERNAL  DOPL
      INTEGER*4 DOPL
C       DoPolymarker            returns DtObject
      EXTERNAL  DOPM
      INTEGER*4 DOPM
C       DoPrimSurf              returns DtObject
      EXTERNAL  DOPMS
      INTEGER*4 DOPMS
C       DoPointList             returns DtObject
      EXTERNAL  DOPNTL
      INTEGER*4 DOPNTL
C       DoPopAtts               returns DtObject
      EXTERNAL  DOPPA
      INTEGER*4 DOPPA
C       DoPopMatrix             returns DtObject
      EXTERNAL  DOPPMX
      INTEGER*4 DOPPMX
C       DoProjection            returns DtObject
      EXTERNAL  DOPRJ
      INTEGER*4 DOPRJ
C       DoPickSwitch            returns DtObject
      EXTERNAL  DOPS
      INTEGER*4 DOPS
C       DoPushAtts              returns DtObject
      EXTERNAL  DOPUA
      INTEGER*4 DOPUA
C       DoPushMatrix            returns DtObject
      EXTERNAL  DOPUMX
      INTEGER*4 DOPUMX
C       DoRaster                returns DtObject
      EXTERNAL  DORS
      INTEGER*4 DORS
C       DoRasterFile            returns DtObject
      EXTERNAL  DORF
      INTEGER*4 DOR
C       DoRasterData            returns DtObject
      EXTERNAL  DORD
      INTEGER*4 DORD
C       DoReflectionSwitch      returns DtObject
      EXTERNAL  DOREFS
```

```
      INTEGER*4 DOREFS
C       DoRefractionSwitch    returns DtObject
      EXTERNAL  DORFRS
      INTEGER*4 DORFRS
C       DoRefractionIndex     returns DtObject
      EXTERNAL  DORFRI
      INTEGER*4 DORFRI
C       DoRepType             returns DtObject
      EXTERNAL  DOREPT
      INTEGER*4 DOREPT
C       DoRotate              returns DtObject
      EXTERNAL  DOROT
      INTEGER*4 DOROT
C       DoSampleAdaptive      returns DtObject
      EXTERNAL  DOSADP
      INTEGER*4 DOSADP
C       DoSampleAdaptiveSwitch  returns DtObject
      EXTERNAL  DOSASW
      INTEGER*4 DOSASW
C       DoSampleFilter          returns DtObject
      EXTERNAL  DOSFLT
      INTEGER*4 DOSFLT
C       DoSampleJitter          returns DtObject
      EXTERNAL  DOSJIT
      INTEGER*4 DOSJIT
C       DoSampleJitterSwitch    returns DtObject
      EXTERNAL  DOSJSW
      INTEGER*4 DOSJSW
C       DoSampleSuper           returns DtObject
      EXTERNAL  DOSSPR
      INTEGER*4 DOSSPR
C       DoSampleSuperSwitch     returns DtObject
      EXTERNAL  DOSSSW
      INTEGER*4 DOSSSW
C       DoScale               returns DtObject
      EXTERNAL  DOSC
      INTEGER*4 DOSC
C       DoSubDivSpec          returns DtObject
      EXTERNAL  DOSDS
      INTEGER*4 DOSDS
C       DoShadowSwitch        returns DtObject
      EXTERNAL  DOSHAS
      INTEGER*4 DOSHAS
C       DoShear               returns DtObject
      EXTERNAL  DOSHR
      INTEGER*4 DOSHR
C       DoShadeIndex          returns DtObject
      EXTERNAL  DOSI
      INTEGER*4 DOSI
C       DoSpecularColor       returns DtObject
      EXTERNAL  DOSPCC
      INTEGER*4 DOSPCC
C       DoSpecularFactor      returns DtObject
      EXTERNAL  DOSPCF
      INTEGER*4 DOSPCF
C       DoSpecularIntens      returns DtObject
```

```
      EXTERNAL  DOSPCI
      INTEGER*4 DOSPCI
C       DoSpecularSwitch      returns DtObject
      EXTERNAL  DOSPCS
      INTEGER*4 DOSPCS
C       DoSimplePolygon       returns DtObject
      EXTERNAL  DOSPGN
      INTEGER*4 DOSPGN
C       DoSimplePolygonMesh   returns DtObject
      EXTERNAL  DOSPM
      INTEGER*4 DOSPM
C       DoSurfaceShade        returns DtObject
      EXTERNAL  DOSRFS
      INTEGER*4 DOSRFS
C       DoStereo              returns DtObject
      EXTERNAL  DOSTER
      INTEGER*4 DOSTER
C       DoStereoSwitch        returns DtObject
      EXTERNAL  DOSTES
      INTEGER*4 DOSTES
C       DoTextAlign           returns DtObject
      EXTERNAL  DOTA
      INTEGER*4 DOTA
C       DoTranspColor         returns DtObject
      EXTERNAL  DOTC
      INTEGER*4 DOTC
C       DoTextExpFactor       returns DtObject
      EXTERNAL  DOTEF
      INTEGER*4 DOTEF
C       DoTextFont            returns DtObject
      EXTERNAL  DOTF
      INTEGER*4 DOTF
C       DoTextHeight          returns DtObject
      EXTERNAL  DOTH
      INTEGER*4 DOTH
C       DoTranspIntens        returns DtObject
      EXTERNAL  DOTI
      INTEGER*4 DOTI
C       DoTransformMatrix     returns DtObject
      EXTERNAL  DOTMX
      INTEGER*4 DOTMX
C       DoTorus               returns DtObject
      EXTERNAL  DOTOR
      INTEGER*4 DOTOR
C       DoTextPath            returns DtObject
      EXTERNAL  DOTPA
      INTEGER*4 DOTPA
C       DoTextPrecision       returns DtObject
      EXTERNAL  DOTPR
      INTEGER*4 DOTPR
C       DoTriangleList        returns DtObject
      EXTERNAL  DOTRIL
      INTEGER*4 DOTRIL
C       DoTriangleMesh        returns DtObject
      EXTERNAL  DOTRIM
      INTEGER*4 DOTRIM
```

```
C       DoTranspSwitch          returns DtObject
      EXTERNAL  DOTS
      INTEGER*4 DOTS
C       DoTranspOrientColor     returns DtObject
      EXTERNAL  DOTOC
      INTEGER*4 DOTOC
C       DoTranspOrientIntens    returns DtObject
      EXTERNAL  DOTOI
      INTEGER*4 DOTOI
C       DoTranspOrientExp       returns DtObject
      EXTERNAL  DOTOE
      INTEGER*4 DOTOE
C       DoTranspOrientSwitch    returns DtObject
      EXTERNAL  DOTOS
      INTEGER*4 DOTOS
C       DoTextSpace             returns DtObject
      EXTERNAL  DOTSP
      INTEGER*4 DOTSP
C       DoTextUpVector          returns DtObject
      EXTERNAL  DOTUV
      INTEGER*4 DOTUV
C       DoText                  returns DtObject
      EXTERNAL  DOTXT
      INTEGER*4 DOTXT
C       DoTextureAntiAlias        returns DtObject
      EXTERNAL  DOTAA
      INTEGER*4 DOTAA
C       DoTextureExtendUV         returns DtObject
      EXTERNAL  DOTXUV
      INTEGER*4 DOTXUV
C       DoTextureExtendUVW        returns DtObject
      EXTERNAL  DOTXW
      INTEGER*4 DOTXW
C       DoTextureMapBump          returns DtObject
      EXTERNAL  DOTMB
      INTEGER*4 DOTMB
C       DoTextureMapBumpSwitch    returns DtObject
      EXTERNAL  DOTMBS
      INTEGER*4 DOTMBS
C       DoTextureMapDiffuseColor returns DtObject
      EXTERNAL  DOTMDC
      INTEGER*4 DOTMDC
C       DoTextureMapDiffuseColorSwitch returns DtObject
      EXTERNAL  DOTMDS
      INTEGER*4 DOTMDS
C       DoTextureMapEnviron       returns DtObject
      EXTERNAL  DOTME
      INTEGER*4 DOTME
C       DoTextureMapEnvironSwitch returns DtObject
      EXTERNAL  DOTMES
      INTEGER*4 DOTMES
C       DoTextureMapTranspIntens returns DtObject
      EXTERNAL  DOTMTI
      INTEGER*4 DOTMTI
C       DoTextureMapTranspIntensSwitch returns DtObject
      EXTERNAL  DOTMTS
```

```
      INTEGER*4 DOTMTS
C       DoTextureOp              returns DtObject
      EXTERNAL  DOTOP
      INTEGER*4 DOTOP
C       DoTextureScaleUV         returns DtObject
      EXTERNAL  DOTSUV
      INTEGER*4 DOTSUV
C       DoTextureScaleUVW        returns DtObject
      EXTERNAL  DOTSW
      INTEGER*4 DOTSW
C       DoTextureTranslateUV     returns DtObject
      EXTERNAL  DOTTUV
      INTEGER*4 DOTTUV
C       DoTextureTranslateUVW    returns DtObject
      EXTERNAL  DOTTW
      INTEGER*4 DOTTW
C       DoTextureUVIndex         returns DtObject
      EXTERNAL  DOTUVI
      INTEGER*4 DOTUVI
C       DoTextureUVWIndex        returns DtObject
      EXTERNAL  DOTWI
      INTEGER*4 DOTWI
C       DoVarTriangleMesh      returns DtObject
      EXTERNAL  DOVTRM
      INTEGER*4 DOVTRM
C       DoVarSimplePolygonMesh      returns DtObject
      EXTERNAL  DOVSPM
      INTEGER*4 DOVSPM
C       DoVarLineList          returns DtObject
      EXTERNAL  DOVLNL
      INTEGER*4 DOVLNL
C       DoVarPointList         returns DtObject
      EXTERNAL  DOVPTL
      INTEGER*4 DOVPTL
C       DoView                 returns DtObject
      EXTERNAL  DOVW
      INTEGER*4 DOVW
C       DoTranslate            returns DtObject
      EXTERNAL  DOXLT
      INTEGER*4 DOXLT
C       DpUpdateVarTriangleMesh returns void
      EXTERNAL  DPUVTM
C       DpUpdateVarSimplePolygonMesh    returns void
      EXTERNAL  DPUVSM
C       DpUpdateVarLineList      returns void
      EXTERNAL  DPUVLL
C       DpUpdateVarPointList     returns void
      EXTERNAL  DPUVPL
C       DsCompBoundingVol      (DtVolume *)
      EXTERNAL  DSCBV
      INTEGER*4 DSCBV
C       DsExecutionAbort       returns void
      EXTERNAL  DSEA
C       DsExecuteObj           returns void
      EXTERNAL  DSEO
C       DsExecutionReturn      returns void
```

```
      EXTERNAL  DSER
C       DsFileRasterRead        returns void
      EXTERNAL  DSFRSR
      INTEGER*4 DSFRSR
C       DsHoldObj               returns DtObject
      EXTERNAL  DSHO
      INTEGER*4 DSHO
C       DsInitializeSystem      returns void
      EXTERNAL  DSINIT
C       DsInputValue            returns void
      EXTERNAL  DSIV
C       DsPrintObj              returns void
      EXTERNAL  DSPO
C       DsInqClassId              returns DtInt
      EXTERNAL  DSQCI
      INTEGER*4 DSQCI
C       DsInqCurrentMethod      returns DtInt
      EXTERNAL  DSQCM
      INTEGER*4 DSQCM
C       DsInqExeDepthLimit      returns DtInt
      EXTERNAL  DSQEDL
      INTEGER*4 DSQEDL
C       DsInqErrorMessage       returns DtReadStatus
      EXTERNAL  DSQEM
      INTEGER*4 DSQEM
C       DsInqErrorVars          returns void
      EXTERNAL  DSQEV
C       DsInqHoldObj            returns DtFlag
      EXTERNAL  DSQHO
      INTEGER*4 DSQHO
C       DsInqMethodId           return DtInt
      EXTERNAL  DSQMI
      INTEGER*4 DSQMI
C       DsInqNumRenderers()     returns DtInt
      EXTERNAL  DSQNF
      INTEGER*4 DSQNF
C       DsInqObj (DcNameInteger, ...)          returns DtObject
      EXTERNAL  DSQOI
      INTEGER*4 DSQOI
C       DsInqObjName of type DcNameInteger     returns DtInt
      EXTERNAL  DSQONI
      INTEGER*4 DSQONI
C       DsInqObjName of type DcNameString      returns void
      EXTERNAL  DSQONS
C       DsInqObjName return name type          returns DtInt
      EXTERNAL  DSQONT
      INTEGER*4 DSQONT
C       DsInqObj (DcNameString, ...)           returns DtObject
      EXTERNAL  DSQOS
      INTEGER*4 DSQOS
C       DsInqObjType            returns DtInt
      EXTERNAL  DSQOT
      INTEGER*4 DSQOT
C       DsInqRendererId         returns DtInt
      EXTERNAL  DSQRI
      INTEGER*4 DSQRI
```

```
C       DsInqRendererNames       returns void
      EXTERNAL  DSQRNS
      INTEGER*4 DSQRNS
C       DsInqSafeFlag          returns DtFlag
      EXTERNAL  DSQSF
      INTEGER*4 DSQSF
C       DsInqValuatorGroup     returns DtObject
      EXTERNAL  DSQVG
      INTEGER*4 DSQVG
C       DsInqVersion           returns void
      EXTERNAL  DSQVER
C       DsInqObjStatus         returns DtObjectStatus
      EXTERNAL  DSQVOS
      INTEGER*4 DSQVOS
C       DsRasterUpdate           returns void
      EXTERNAL  DSRSU
C       DsRasterWrite            returns void
      EXTERNAL  DSRSW
C       DsReleaseObj           returns void
      EXTERNAL  DSRO
C       DsSetExeDepthLimit     returns void
      EXTERNAL  DSSEDL
C       DsSetErrorVars         returns void
      EXTERNAL  DSSEV
C       DsSetObjName (..., DcNameNone, ...)           returns void
      EXTERNAL  DSSOND
C       DsSetObjName (..., DcNameInteger, ...)        returns void
      EXTERNAL  DSSONI
C       DsSetObjName (..., DcNameString, ...)         returns void
      EXTERNAL  DSSONS
C       DsSetSafeFlag          returns void
      EXTERNAL  DSSSF
C       DsTerminateSystem      returns void
      EXTERNAL  DSTERM
C       DsTextureUVCount         returns DtInt
      EXTERNAL  DSTUVC
      INTEGER*4 DSTUVC
C       DsTextureUVWCount        returns DtInt
      EXTERNAL  DSTWC
      INTEGER*4 DSTWC
C       DsUpdateAllViews       returns void
      EXTERNAL  DSUAV
C       DsValuatorSwitch       returns void
      EXTERNAL  DSVS
C       DvInqActiveCamera      returns DtObject
      EXTERNAL  DVQAC
      INTEGER*4 DVQAC
C       DvInqBoundary          returns void
      EXTERNAL  DVQB
C       DvInqBackgroundColor   returns void
      EXTERNAL  DVQBC
C       DvInqClearFlag         returns DtFlag
      EXTERNAL  DVQCF
      INTEGER*4 DVQCF
C       DvInqDefinitionGroup   returns DtObject
      EXTERNAL  DVQDG
```

```
      INTEGER*4 DVQDG
C       DvInqDisplayGroup     returns DtObject
      EXTERNAL  DVQIG
      INTEGER*4 DVQIG
C       DvInqRendSyle         returns DtRenderStyle
      EXTERNAL  DVQRS
      INTEGER*4 DVQRS
C       DvInqShadeIndex       returns DtInt
      EXTERNAL  DVQSI
      INTEGER*4 DVQSI
C       DvInqUpdateType       returns DtUpdateType
      EXTERNAL  DVQUT
      INTEGER*4 DVQUT
C       DvSetActiveCamera     returns void
      EXTERNAL  DVSAC
C       DvSetBoundary         returns void
      EXTERNAL  DVSB
C       DvSetBackgroundColor  returns void
      EXTERNAL  DVSBC
C       DvSetClearFlag        returns void
      EXTERNAL  DVSCF
C       DvSetRendStyle        returns void
      EXTERNAL  DVSRS
C       DvSetShadeIndex       returns void
      EXTERNAL  DVSSI
C       DvSetUpdateType       returns void
      EXTERNAL  DVSUT
C       DvUpdate              returns void
      EXTERNAL  DVU
C       DeAddClass            returns DtInt
      EXTERNAL  DEAC
      INTEGER*4 DEAC
C       DeCreateObject        returns DtObject
      EXTERNAL  DECO
      INTEGER*4 DECO
C       DeDeleteObject        returns void
      EXTERNAL  DEDO
C       DeExecuteAlternate    returns void
      EXTERNAL  DEEA
C       DeInitializeObjPick   returns void
      EXTERNAL  DEIOP
C       DeInqPickable         returns DtFlag
      EXTERNAL  DEQP
      INTEGER*4 DEQP
C       DeInqRenderable       returns DtFlag
      EXTERNAL  DEQR
      INTEGER*4 DEQR
C       Fortran only.   Add method to list     returns void
      EXTERNAL DEAMTH
C       Fortran only.   Destroy object private data     returns void
      EXTERNAL DEDOD
C       Fortran only.   Get object private data     returns void
      EXTERNAL DEROD
C       Fortran only.   Put object private data     returns void
      EXTERNAL DEWOD
```

Filename: DOREMETHODS

```
C************************************************************************
CCopyright (C) 1989, by Stardent Computer Corp.
C        All Rights Reserved
CThis program is a  trade secret of  Stardent Computer Corp. and it is not
Cto be reproduced, published, disclosed  to others, copied, adapted,
Cdistributed, or displayed without the prior authorization of Stardent
CComputer Corp.  Licensee agrees  to  attach  or  embed  this  Notice on
Call copies  of the program, including partial copies or modified versions
Cthereof.
C************************************************************************
C
C**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**
C**                                                                      **
C**  This file contains all the Dore FORTRAN method constant declarations. **
C**                                                                      **
C************************************************************************

C methods

C DcMethodNull
      INTEGER*4 DCMNUL
      PARAMETER (DCMNUL =-1)
C DcMethodDestroy
      INTEGER*4 DCMDST
      PARAMETER (DCMDST =0)
C DcMethodAddReference
      INTEGER*4 DCMADR
      PARAMETER (DCMADR =1)
C DcMethodRemoveReference
      INTEGER*4 DCMRMR
      PARAMETER (DCMRMR =2)
C DcMethodPrint
      INTEGER*4 DCMPRT
      PARAMETER (DCMPRT =3)
C DcMethodCheckGroup
      INTEGER*4 DCMCKG
      PARAMETER (DCMCKG =4)
C DcMethodCmpBndVolume
      INTEGER*4 DCMCBV
      PARAMETER (DCMCBV =5)
C DcMethodIniPick
      INTEGER*4 DCMINP
      PARAMETER (DCMINP =6)
C DcMethodPick
      INTEGER*4 DCMPCK
      PARAMETER (DCMPCK =7)
C DcMethodInqGlbAttVal
      INTEGER*4 DCMIGA
      PARAMETER (DCMIGA =8)
```

```
C DcMethodUpdStdAltObj
      INTEGER*4 DCMSAO
      PARAMETER (DCMSAO =9)
C DcMethodStdRenderStudio
      INTEGER*4 DCMSRS
      PARAMETER (DCMSRS =10)
C DcMethodStdRenderDisplay
      INTEGER*4 DCMSRD
      PARAMETER (DCMSRD =11)

C DcMethodDynIniRender
      INTEGER*4 DCMDIR
      PARAMETER (DCMDIR =12)
C DcMethodDynRender
      INTEGER*4 DCMDR
      PARAMETER (DCMDR =13)
C DcMethodGlbrndIniRender
      INTEGER*4 DCMGIR
      PARAMETER (DCMGIR =14)
C DcMethodGlbrndIniObjs
      INTEGER*4 DCMGIO
      PARAMETER (DCMGIO =15)
C DcMethodGlbrndSpcboxOvr
      INTEGER*4 DCMGSO
      PARAMETER (DCMGSO =16)
C DcMethodGlbrndRayint
      INTEGER*4 DCMGRI
      PARAMETER (DCMGRI =17)
C DcMethodGlbrndUsrdat
      INTEGER*4 DCMGUD
      PARAMETER (DCMGUD =18)
C DcMethodGlbrndWcsloc
      INTEGER*4 DCMGWC
      PARAMETER (DCMGWC =19)
C DcMethodGlbrndWcsnrm
      INTEGER*4 DCMGWM
      PARAMETER (DCMGWM =20)
C DcMethodGlbrndWldBnd
      INTEGER*4 DCMGWB
      PARAMETER (DCMGWB =21)
```

Filename: DORETEXT

```
C************************************************************************
CCopyright (C) 1989, by Stardent Computer Corp.
C        All Rights Reserved
CThis program is a  trade secret of  Stardent Computer Corp. and it is not
Cto be reproduced, published, disclosed  to others, copied, adapted,
Cdistributed, or displayed without the prior authorization of Stardent
CComputer Corp.  Licensee agrees  to  attach  or  embed  this  Notice on
Call copies  of the program, including partial copies or modified versions
Cthereof.
C************************************************************************
C
C**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**
C**                                                                      **
C**  This file contains all the Dore FORTRAN text constant declarations.  **
C**                                                                      **
C************************************************************************

C DtTextPrecision;
C       DcStringPrecision
      INTEGER*4 DCSTRP
      PARAMETER (DCSTRP =0)
C       DcCharacterPrecision
      INTEGER*4 DCCHRP
      PARAMETER (DCCHRP =1)
C       DcStrokePrecision
      INTEGER*4 DCSTKP
      PARAMETER (DCSTKP =2)

C DtTextAlignHorizontal;
C       DcTextHAlignNormal
      INTEGER*4 DCTHAN
      PARAMETER (DCTHAN =0)
C       DcTextHAlignLeft
      INTEGER*4 DCTHAL
      PARAMETER (DCTHAL =1)
C       DcTextHAlignCenter
      INTEGER*4 DCTHAC
      PARAMETER (DCTHAC =2)
C       DcTextHAlignRight
      INTEGER*4 DCTHAR
      PARAMETER (DCTHAR =3)

C DtTextAlignVertical;
C       DcTextVAlignNormal
      INTEGER*4 DCTVAN
      PARAMETER (DCTVAN =0)
C       DcTextVAlignTop
      INTEGER*4 DCTVAT
      PARAMETER (DCTVAT =1)
```

```
C       DcTextVAlignCap
      INTEGER*4 DCTVAC
      PARAMETER (DCTVAC =2)
C       DcTextVAlignHalf
      INTEGER*4 DCTVAH
      PARAMETER (DCTVAH =3)
C       DcTextVAlignBase
      INTEGER*4 DCTVAB
      PARAMETER (DCTVAB =4)
C       DcTextVAlignBottom
      INTEGER*4 DCTVAM
      PARAMETER (DCTVAM =5)

C DtTextPath;
C       DcTextPathRight
      INTEGER*4 DCTPR
      PARAMETER (DCTPR =0)
C       DcTextPathLeft
      INTEGER*4 DCTPL
      PARAMETER (DCTPL =1)
C       DcTextPathUp
      INTEGER*4 DCTPU
      PARAMETER (DCTPU =2)
C       DcTextPathDown
      INTEGER*4 DCTPD
      PARAMETER (DCTPD =3)

C DtFont;
C       DcPlainRoman
      INTEGER*4 DCFPR
      PARAMETER (DCFPR =0)
C       DcSimplexRoman
      INTEGER*4 DCFSR
      PARAMETER (DCFSR =1)
C       DcDuplexRoman
      INTEGER*4 DCFDR
      PARAMETER (DCFDR =2)
C       DcComplexSmallRoman
      INTEGER*4 DCFCSR
      PARAMETER (DCFCSR =3)
C       DcComplexRoman
      INTEGER*4 DCFCR
      PARAMETER (DCFCR =4)
C       DcTriplexRoman
      INTEGER*4 DCFTR
      PARAMETER (DCFTR =5)
C       DcComplexSmallItalic
      INTEGER*4 DCFCSI
      PARAMETER (DCFCSI =6)
C       DcComplexItalic
      INTEGER*4 DCFCI
      PARAMETER (DCFCI =7)
C       DcTriplexItalic
      INTEGER*4 DCFTI
      PARAMETER (DCFTI =8)
C       DcSimplexScript
```

```
      INTEGER*4 DCFSS
      PARAMETER (DCFSS =9)
C       DcComplexScript
      INTEGER*4 DCFCS
      PARAMETER (DCFCS =10)
C       DcGothicGerman
      INTEGER*4 DCFGG
      PARAMETER (DCFGG =11)
C       DcGothicEnglish
      INTEGER*4 DCFGE
      PARAMETER (DCFGE =12)
C       DcGothicItalian
      INTEGER*4 DCFGI
      PARAMETER (DCFGI =13)
C       DcPlainGreek
      INTEGER*4 DCFPG
      PARAMETER (DCFPG =14)
C       DcSimplexGreek
      INTEGER*4 DCFSG
      PARAMETER (DCFSG =15)
C       DcComplexSmallGreek
      INTEGER*4 DCFCSG
      PARAMETER (DCFCSG =16)
C       DcComplexGreek
      INTEGER*4 DCFCG
      PARAMETER (DCFCG =17)
C       DcComplexCyrillic
      INTEGER*4 DCFCC
      PARAMETER (DCFCC =18)
C       DcUpperCaseMathematics
      INTEGER*4 DCFUCM
      PARAMETER (DCFUCM =19)
C       DcLowerCaseMathematics
      INTEGER*4 DCFLCM
      PARAMETER (DCFLCM =20)
C       DcMusic
      INTEGER*4 DCFMUS
      PARAMETER (DCFMUS =21)
C       DcMeteorology
      INTEGER*4 DCFMET
      PARAMETER (DCFMET =22)
C       DcSymbols
      INTEGER*4 DCFSYM
      PARAMETER (DCFSYM =23)
C       DcAstrology
      INTEGER*4 DCFAST
      PARAMETER (DCFAST =24)
C       DcHelvetica
      INTEGER*4 DCFHLV
      PARAMETER (DCFHLV =25)
```

QUICK REFERENCE

CHAPTER FOUR

This chapter provides an alphabetical listing of the Doré functions. A similar listing, arranged by function type, is located in Chapter Three of the *Doré Programmer's Guide*. C (mixed case) functions are listed first; an alphabetical Fortran (all upper case) cross reference list begins on page 4-31.

DdInqColorEntries (device, colormodel, start, count, entries)
DDQCE (DEVICE, COLMOD, START, COUNT, ENTRYS)
Return color lookup table entries of a device

DdInqColorTableSize (device)
DDQCTS (DEVICE)
Return the number of entries in the color lookup table

DdInqExtent (device, volume)
DDQE (DEVICE, VOLUME)
Return the extent of a device

DdInqFonts (device, fonts)
DDQFT (DEVICE, FONTS)
Return the set of fonts supported by a device

DdInqFrame (device)
DDQFR (DEVICE)
Return the frame of a device

DdInqNumFonts (device)
DDQNF (DEVICE)
Return the number of fonts supported by a device

DdInqPickAperture (device, aperture)
DDQPA (DEVICE, APRTUR)
Return the pick aperture of a device

DdInqPickCallback (device)
DDQPC (DEVICE)
Return the pick callback object of a device

DdInqPickPathOrder (device)
DDQPPO (DEVICE)
Return the pick path order of a device

DdInqPixelData (device, requesttype, width, height, type, data, userdelete)
DDQPXD (DEVICE, REQTYP, WIDTH, HEIGHT, TYPE, DATA, USRDEL)
Return pixel information about an image on a given device

DdInqResolution (device, x, y)
DDQR (DEVICE, X, Y)
Return the resolution of a device

DdInqShadeMode (device)
DDQSM (DEVICE)
Return the shade mode of a device

DdInqShadeRanges (device, start, count, entries)
DDQSR (DEVICE, START, COUNT, ENTRYS)
Return the shade range table entries of a device

DdInqViewport (device, viewport)
DDQV (DEVICE, VWPORT)
Return the size of the device viewport of a device

DdInqVisualType (device)
DDQVT (DEVICE)
Return the visual type of a device

DdPickObjs (device, pick_point, hit_count, index_size, index, list_size, hit_list, z_values, wcs_values, lcs_values, views, error_word)
DDPO (DEVICE, PICKPT, HITCNT, IDXSIZ, INDEX, LSTSIZ, HITLST, ZVALUS, WCSVAL, LCSVAL, VIEWS, ERRWRD)
Initiate a pick on a device

DdSetColorEntries (device, colormodel, start, count, entries)
DDSCE (DEVICE, COLMOD, START, COUNT, ENTRYS)
Set the color lookup table entries of a device

DdSetFrame (device, frame)
DDSF (DEVICE, FRAME)
Attach a frame to a device

DdSetPickAperture (device, aperture)
DDSPA (DEVICE, APRTUR)
Set the pick aperture of a device

DdSetPickCallback (device, pickcallbackobj)
DDSPCB (DEVICE, PKCBOJ)
Set the picking callback object of a device

DdSetPickPathOrder (device, pathorder)
DDSPPO (DEVICE, PTHORD)
Set the order of pick path elements returned to *DdPickObjs*

DdSetShadeMode (device, mode)
DDSSM (DEVICE, MODE)
Set the shade mode of a device

DdSetShadeRanges (device, start, count, entries)
DDSSR (DEVICE, START, COUNT, ENTRYS)
Set one or more shade range table entries of a device

DdSetViewport (device, viewport)
DDSDV (DEVICE, VWPORT)
Define a device viewport for a device

DdUpdate (device)
DDU (DEVICE)
Update the specified device

DEOD (OBJECT)
Deallocate space used by the private data of an object of a user-defined class (Fortran only)

DEROD (OBJECT, TODATA, SIZE)
Read private data of an object of a user-defined class (Fortran only)

DEWOD (OBJECT, FROMDATA, SIZE)
Write private data of an object of a user-defined class (Fortran only)

DeAddClass (name, count, list, default_routine)
DEAC (NAME, N, COUNT, LIST, DFLTRT)
Add a new class (object type)

DEAMTH (LIST, METHOD, RTN, MTHCNT)
Build a method list (Fortran only – used with DEAC)

DeCreateObject (class_id, object_data)
DECO (CLASID, DATA, SIZE)
Create an internal Doré object

DeDeleteObject (object)
DEDO (OBJECT)
Delete a Doré object

DeExecuteAlternate (object)
DEEA (OBJECT)
Execute the current method on an alternate object

DeInitializeObjPick (object)
DEIOP (OBJECT)
Initialize picking for an object

DeInqPickable (class_id)
DEQP (CLSID)
Return whether a class is pickable

DeInqRenderable (class_id)
DEQR (CLSID)
Return whether a class is renderable

DfInqBoundary (frame, boundary)
DFQB (FRAME, BNDRY)
Return the frame boundary

DfInqJust (frame, left, bottom)
DFQJ (FRAME, LEFT, BOTTOM)
Return the frame justification

DfInqViewGroup (frame)
DFQVG (FRAME)
Return the handle for a frame's view group

DfSetBoundary (frame, boundary)
DFSB (FRAME, BNDRY)
Set the frame boundary

DfSetJust (frame, left, bottom)
DFSJ (FRAME, LEFT, BOTTOM)
Set the frame justification

DfUpdate (frame)
DFU (FRAME)
Update the specified frame

DgAddObj (object)
DGAO (OBJECT)
Add an object to the currently open group

DgAddObjToGroup (group, object)
DGAOG (GROUP, OBJECT)
Add an object to a specified group

DgCheck (group)
DGCK (GROUP)
Check for circularities within a group network

DgClose ()
DGCS ()
Close a group object

DgDelEle (count)
DGDE (COUNT)
Remove elements from the currently open group

DgDelEleBetweenLabels (label1, label2)
DGDEL (LABEL1, LABEL2)
Remove all elements between labels from the currently open group

DgDelEleRange (from, to)
DGDER (FROM, TO)
Remove a range of elements from the currently open group

DgEmpty (group)
DGE (GROUP)
Remove all elements from a specified group

DgInqElePtr ()
DGQEP ()
Return the location of the group element pointer of the current group

DgInqObjAtPos (group, offset, type)
DGQOP (GROUP, OFFSET, TYPE)
Return the object at a specified position in a group

DgInqOpen ()
DGQO ()
Determine which, if any, group is open

DgInqSize (group)
DGQS (GROUP)
Return the number of elements in the specified group

DgOpen (group, append)
DGO (GROUP, APPEND)
Open a group object

DgReplaceObj (object)
DGRO (OBJECT)
Replace an object in the currently open group

DgReplaceObjInGroup (group, object)
DGROG (GROUP, OBJECT)
Replace an object in a specified group

DgSetElePtr (element_ptr, type)
DGSEP (ELEPTR, TYPE)
Set the group element pointer within the current group

DgSetElePtrRelLabel (label, offset)
DGSEPL (LABEL, OFFSET)
Set the group element pointer relative to a label

DoAmbientIntens (intensity)
DOAMBI (INTENS)
Create an ambient intensity primitive attribute object

DoAmbientSwitch (switchvalue)
DOAMBS (SWVAL)
Create an ambient switch primitive attribute object

DoAnnoText (position, string)
DOANNT (POSITN, STRING, N)
Create an annotation text primitive object

DoBackfaceCullSwitch (switchvalue)
DOBFCS (SWVAL)
Create a primitive attribute object for enabling/disabling backface culling

DoBackfaceCullable (switchvalue)
DOBFC (SWVAL)
Create a primitive attribute object defining backface cullability

DoBoundingVol (volume, alternate_object)
DOBV (VOLUME, ALTOBJ)
Create a bounding volume object

DoBoundingVolSwitch (switchvalue)
DOBVS (SWVAL)
Create a bounding volume switch primitive attribute object

DoCallback (function,dataobject)
DOCB (FUNCT, DATOBJ)
Create a callback object

DoCamera ()
DOCM ()
Create a camera studio object

DoCameraMatrix (matrix)
DOCMX (MATRIX)
Create a camera matrix studio attribute object

DoClipSwitch (switchvalue)
DOCS (SWVAL)
Create a primitive attribute object for enabling/disabling model clipping

DoClipVol (operator, numhalfspaces, halfspaces)
DOCV (OPRATR, NHLFSP, HLFSPS)
Create a primitive attribute object defining a model clipping volume

DoDataPtr (dataptr)
DODP (DATPTR)
Create a data pointer object

DoDataVal (dataval)
DODV (DATVAL)
Create a data value object

DoDepthCue (zfront, zback, sfront, sback, colormodel, color)
DODC (ZFRONT, ZBACK, SFRONT, SBACK, COLMOD, COLOR)
Create a primitive attribute object defining depth cueing

DoDepthCueSwitch (switchvalue)
DODCS (SWVAL)
Create a primitive attribute object for enabling/disabling depth cueing

DoDevice (devicetype, argstring)
DOD (DEVTYP, LDEV, ARGSTR, LARG)
Open a device

DoDiffuseColor (colormodel, color)
DODIFC (COLMOD, COLOR)
Create a diffuse color primitive attribute object

DoDiffuseIntens (intensity)
DODIFI (INTENS)
Create a diffuse intensity primitive attribute object

DoDiffuseSwitch (switchvalue)
DODIFS (SWVAL)
Create a diffuse switch primitive attribute object

DoExecSet (n, list, setop)
DOES (N, LIST, SETOP)
Create an object that modifies the executability of objects

DoFileRaster (filename, specialstring)
DOFRS (FNAME, FLEN, SPCSTR, SLEN)
Create a fileraster object

DoFilter (filter, n, members, setop)
DOFL (FILTER, N, MEMBER, SETOP)
Create a filter modifying object

DoFrame ()
DOFR ()
Create a frame object

DoGenerateTextureUV (switchvalue)
DOGTUV (SWVAL)
Create a primitive attribute object for enabling/disabling generation of **uv** texture coordinates

DoGlbRendMaxObjs (maxobjs)
DOGRMO (MAXOBJ)
Create a studio attribute object that defines the maximum number of objects per spatial subdivision for global ray casting

DoGlbRendMaxSub (maxsub)
DOGRMS (MAXSUB)
Create a studio attribute object that defines the maximum number of subdivisions for global ray casting

DoGlbRendRayLevel (raylevel)
DOGRRL (RAYLEV)
Create a ray level studio attribute object

DoGroup (open)
DOG (OPEN)
Create a group organizational object

DoHiddenSurfSwitch (switchvalue)
DOHSS (SWVAL)
Create a hidden surface switch primitive attribute object

DoInLineGroup (open)
DOILG (OPEN)
Create an inline group object

DoInputSlot ()
DOIS ()
Create an input slot organizational object

DoInterpType (type)
DOIT (TYPE)
Create an interpolation type primitive attribute object

DoInvisSwitch (switchvalue)
DOINVS (SWVAL)
Create an invisibility switch primitive attribute object

DoLabel (label)
DOLL (LABEL)
Create a label organization object

DoLight ()
DOLT ()
Create a light studio object

DoLightAttenuation (c1, c2)
DOLTA (C1, C2)
Create a light attribute object defining light intensity falloff with distance

DoLightColor (colormodel, color)
DOLC (COLMOD, COLOR)
Create a light color studio attribute object

DoLightIntens (intensity)
DOLI (INTENS)
Create a light intensity studio attribute object

DoLightSpreadAngles (total_angle, delta_angle)
DOLTSA (TANGLE,DANGLE)
Create a light attribute defining the width of the light beam

DoLightSpreadExp (exponent)
DOLSE (EXPN)
Create a light attribute object defining light intensity falloff with angle

DoLightSwitch (light, switchvalue)
DOLTS (LIGHT, SWVAL)
Create a primitive attribute object enabling/disabling illumination from a light

DoLightType (type)
DOLTT (TYPE)
Create a light type studio attribute object

DoLineList (colormodel, vertextype, linecount, vertices)
DOLINL (COLMOD, VTXTYP, LINCNT, VTXS)
Create a line list primitive object

DoLineType (type)
DOLNT (TYPE)
Create a line type primitive attribute object

DoLineWidth (linewidth)
DOLW (LINWID)
Create a line width primitive attribute object

DoLookAtFrom (at, from, up)
DOLAF (AT, FROM, UP)
Create an orienting geometric transformation object

DoMarkerFont (font)
DOMF (FONT)
Create a marker font primitive attribute object

DoMarkerGlyph (glyph)
DOMG (GLYPH)
Create a marker glyph primitive attribute object

DoMarkerScale (scale)
DOMS (SCALE)
Create a marker scale factor primitive attribute object

DoMatrix (n,m,data)
DOM (N, M, DATA)
Create a matrix object

DoMinBoundingVolExt (Extension)
DOMBVE (EXTENS)
Create an attribute object which specifies the minimum renderable bounding extent

DoNURBSurf (colormodel, ctrlpointtype, uv_area, order_u, n_knot_u, knot_u, order_v, n_knot_v, knot_v, n_ctrl_u, n_ctrl_v, ctrl_vertices)
DONRBS (COLMOD, CPTTYP, UVAREA, ORDERU, NKNOTU, KNOTU, ORDERV, NKNOTV, KNOTV, NCTRLU, NCTRLV, CTLVTX)
Create a non-uniform rational B-Spline surface primitive object

DoNameSet (n, members, setop)
DONS (N, MEMBER, SETUP)
Create a nameset modifying object

DoParallel (window_size, hither, yon)
DOPAR (WWSIZE, HITHER, YON)
Create a parallel studio attribute object for cameras

DoPatch (colormodel, vertextype, matrixl, vertices, matrixr)
DOPAT (COLMOD, VTXTYP, MATRXL, VTXS, MATRXR)
Create a patch primitive object

DoPerspective (fov, hither, yon)
DOPER (FOV, HITHER, YON)
Create a perspective studio attribute object for cameras

DoPickID (pickid)
DOPID (PICKID)
Create a pick identifier object

DoPickSwitch (switchvalue)
DOPS (SWVAL)
Create a pickability switch primitive attribute object

DoPointList (colormodel, vertextype, pointcount, vertices)
DOPNTL (COLMOD, VTXTYP, PNTCNT, VTXS)
Create a point list primitive object

DoPolygon (colormodel, vertextype, contourcount, contours, vertices, shape)
DOPGN (COLMOD, VTXTYP, CNTCNT, CONTURS, VTXS, SHAPE)
Create a polygon primitive object

DoPolygonMesh (colormodel, vertextype, vertexcount, vertices, polygoncount, polygons, contours, vertexlist, shape, smoothflag)
DOPGNM (COMOD, VTXTYP, VTXCNT, VTXS, PLYCNT, PLYGON, CONTURS, VTXLST, SHAPE, SMOOFL)
Create a polygon mesh primitive object

DoPolyline (colormodel, vertextype, vertexcount, vertices)
DOPL (COLMOD, VTXTYP, VTXCNT, VTXS)
Create a connected line segment primitive object

DoPolymarker (marker_count, marker_positions)
DOPM (MKRCNT, MKRPOS)
Create a polymarker primitive object

DoPopAtts ()
DOPPA ()
Create an organization object that pops all attributes

DoPopMatrix ()
DOPPMX ()
Create an organization object that pops a matrix

DoPrimSurf (surfacetype)
DOPMS (SRFTYP)
Create a primitive surface primitive object

DoProjection (window, ptype, prp, viewplane, hither, yon)
DOPRJ (WINDOW, PTYPE, PRP, VWPLAN, HITHER, YON)
Create a projection studio attribute object for cameras

DoPushAtts ()
DOPUA ()
Create an organization object that pushes all attributes

DoPushMatrix ()
DOPUMX ()
Create an organization object that pushes a matrix

DoRaster (width, height, depth, type, typestring, data, delcallback)
DORS (WIDTH, HEIGHT, DEPTH, TYPE, TYPSTR, TLEN, DATA, DELCBK)
Create a raster object

DoReflectionSwitch (switchvalue)
DOREFS (SWVAL)
Create a reflection switch primitive attribute object

DoRefractionIndex (index)
DORFRI (INDEX)
Create a refraction index primitive attribute object

DoRefractionSwitch (switchvalue)
DORFRS (SWVAL)
Create a refraction switch primitive attribute object

DoRepType (type)
DOREPT (TYPE)
Create a representation type primitive attribute object

DoRotate (axis, radians)
DOROT (AXIS, RADIAN)
Create a rotation geometric transformation object

DoSampleAdaptive (variance)
DOSADP (VARNCE)
Create an adaptive sampling camera attribute object for antialiasing

DoSampleAdaptiveSwitch (switchvalue)
DOSASW (SWVAL)

Create an adaptive sampling switch camera attribute object for antialiasing

DoSampleFilter (filter, xwidth, ywidth)
DOSFLT (FILTER, XWIDTH, YWIDTH)

Create a supersampling filter camera attribute object for antialiasing

DoSampleJitter (factor)
DOSJIT (FACTOR)

Create a jitter sampling camera attribute object for antialiasing

DoSampleJitterSwitch (switchvalue)
DOSJSW (SWVAL)

Create a jitter sampling switch camera attribute object for antialiasing

DoSampleSuper (xsamples, ysamples)
DOSSPR (XSAMP, YSAMP)

Create a supersampling camera attribute object for antialiasing

DoSampleSuperSwitch (switchvalue)
DOSSSW (SWVAL)

Create a supersampling switch camera attribute object for antialiasing

DoScale (x,y,z)
DOSC (X, Y, Z)

Create a scale geometric tranformation object

DoShadeIndex (index)
DOSI (INDEX)

Create a shade index primitive attribute object

DoShadowSwitch (switchvalue)
DOSHAS (SWVAL)
Create a shadow switch primitive attribute object

DoShear (plane, firstshearvalue, secondshearvalue)
DOSHR (PLANE, D1SHRV, D2SHRV)
Create a shear geometric transformation object

DoSimplePolygon (colormodel, vertextype, vertexcount, vertices, shape)
DOSPGN (COMOD, VTXTYP, VTXCNT, VTXS, SHAPE)
Create a simple polygon primitive object

DoSimplePolygonMesh (colormodel, vertextype, vertexcount, vertices, polygoncount, contours, vertexlist, shape, smoothflag)
DOSPM (COLMOD, VTXTYP, VTXCNT, VTXS, PLYCNT, CONTURS, VTXLST, SHAPE, SMOOFL)
Create a simple polygon mesh primitive object

DoSpecularColor (colormodel, color)
DOSPCC (COLMOD, COLOR)
Create a specular color primitive attribute object

DoSpecularFactor (factor)
DOSPCF (FACTOR)
Create a specular factor primitive attribute object

DoSpecularIntens (intensity)
DOSPCI (INTENS)
Create a specular intensity primitive attribute object

DoSpecularSwitch (switchvalue)
DOSPCS (SWVAL)
Create a specular switch primitive attribute object

DoStereo (eyeseparation, distance)
DOSTER (EYESEP, DISTNC)
Create a stereo studio attribute object

DoStereoSwitch(switchvalue)
DOSTES (SWVAL)
Create a stereo switch studio attribute object

DoSubDivSpec (type, parms)
DOSDS (TYPE, PARMS)
Create a subdivision specification primitive attribute object

DoSurfaceShade (object)
DOSRFS (OBJ)
Create a surface shading primitive attribute object

DoText (position, u, v, string)
DOTXT (POSITN, U, V, STRING, N)
Create a text primitive object

DoTextAlign (halign, valign)
DOTA (HALIGN, VALIGN)
Create a text alignment primitive attribute object

DoTextExpFactor (textexp)
DOTEF (TXEXP)
Create a text expansion factor primitive attribute object

DoTextFont (font)
DOTF (FONT)
Create a text font primitive attribute object

DoTextHeight (textheight)
DOTH (TXHGHT)
Create a text height primitive attribute object

DoTextPath (textpath)
DOTPA (TXPATH)
Create a text path primitive attribute object

DoTextPrecision (precision)
DOTPR (PRECIS)
Create a text precision primitive attribute object

DoTextSpace (textspace)
DOTSP (TXSPAC)
Create a text space primitive attribute object

DoTextUpVector (xup, yup)
DOTUV (XUP, YUP)
Create a text up vector primitive attribute object

DoTextureAntiAlias (mode)
DOTAA (MODE)
Create a texture attribute object that controls antialiasing of textures

DoTextureExtendUV (umode, vmode)
DOTXUV (UMODE, VMODE)
Create a texture attribute object that controls texturing behavior beyond the boundary of a 2-dimensional texture

DoTextureExtendUVW (umode, vmode, wmode)
DOTXW (UMODE, VMODE, WMODE)
Create a texture attribute object that controls texturing behavior beyond the boundary of a 3-dimensional texture

DoTextureMapBump (operator, mapping, raster)
DOTMB (OPRATR, MAPPNG, RASTER)
Create a bump map primitive attribute object

DoTextureMapBumpSwitch (switchvalue)
DOTMBS (SWVAL)
Create a primitive switch attribute object for enabling/disabling bump mapping

DoTextureMapDiffuseColor (operator, mapping, raster)
DOTMDC (OPRATR, MAPPNG, RASTER)

Create a diffuse color texture map primitive attribute object

DoTextureMapDiffuseColorSwitch (switchvalue)
DOTMDS (SWVAL)

Create a primitive attribute object for enabling/disabling texture mapping of diffuse color

DoTextureMapEnviron (operator, mapping, raster)
DOTME (OPRATR, MAPPNG, RASTER)

Create an environment map primitive attribute object

DoTextureMapEnvironSwitch (switchvalue)
DOTMES (SWVAL)

Create a primitive attribute object for enabling/disabling environment mapping

DoTextureMapTranspIntens (operator, mapping, raster)
DOTMTI (OPRATR, MAPPNG, RASTER)

Create a transparent intensity texture map primitive attribute object

DoTextureMapTranspSwitch (switchvalue)
DOTMTS (SWVAL)

Create a primitive attribute object for enabling/disabling texture mapping of transparent intensity

DoTextureOp (operator)
DOTOP (OP)

Create a texture attribute object that controls how the texture value is combined with the current value

DoTextureScaleUV (su, sv)
DOTSUV (SU, SV)

Create a texture attribute object that sets the scale factors for primitive **uv** coordinates

DoTextureScaleUVW (su, sv, sw)
DOTSW (SU, SV, SW)
Create a texture attribute object that sets the scale factors for primitive **uvw** coordinates

DoTextureTranslateUV (tu, tv)
DOTTUV (TU, TV)
Create a texture attribute object that sets the translate factors for primitive **uv** coordinates

DoTextureTranslateUVW (tu, tv, tw)
DOTTW (TU, TV, TW)
Create a texture attribute object that sets the translate factors for primitive **uvw** coordinates

DoTextureUVIndex (index)
DOTUVI (INDEX)
Create a texture attribute object that specifies which **uv** coordinates of a primitive object to use

DoTextureUVWIndex (index)
DOTWI (INDEX)
Create a texture attribute object that specifies which **uvw** coordinates of a primitive object to use

DoTorus (bigradius, smallradius)
DOTOR (BIGRAD, SMLRAD)
Create a torus primitive object

DoTransformMatrix (matrix, comptype)
DOTMX (MATRIX, COMTYP)
Create a transformation matrix geometric transformation object

DoTranslate (x, y, z)
DOXLT (X, Y, Z)
Create a translation geometric transformation object

DoTranspColor (colormodel, color)
DOTC (COLMOD, COLOR)
Create a transparent color primitive attribute object

DoTranspIntens (intensity)
DOTI (INTENS)
Create a transparent intensity primitive attribute object

DoTranspOrientColor (colormodel, color)
DOTOC (COLMOD, COLOR)
Create a transparent color primitive attribute object for orientation dependent transparency

DoTranspOrientExp (exponent)
DOTOE (EXPONENT)
Create a transparent exponent primitive attribute object for orientation dependent transparency

DoTranspOrientIntens (intensity)
DOTOI (INTENS)
Create a transparent intensity primitive attribute object for orientation dependent transparency

DoTranspOrientSwitch (switchvalue)
DOTOS (SWVAL)
Create a primitive attribute object for enabling/ isabling orientation dependent transparency

DoTranspSwitch (switchvalue)
DOTS (SWVAL)
Create a transparent switch primitive attribute object

DoTriangleList (colormodel, vertextype, trianglecount, vertices)
DOTRIL (COLMOD, VTXTYP, TRICNT, VTXS)
Create a triangle list primitive object

DoTriangleMesh (colormodel, vertextype, vertexcount, vertices, trianglecount, triangles, smoothflag)
DOTRIM (COLMOD, VTXTXP, VTXCNT, VTXS, TRICNT, TRIS, SMOOFL)
Create a triangle mesh primitive object

DoVarLineList (colormodel, linecount, vertlocs, vertnorms, vertcolors)
DOVLNL (COLMOD, LINCNT, VTXLOC, VTXNRM, VTXCLR)
Create a variable line list primitive object

DoVarPointList (colormodel, pointcount, vertlocs, vertnorms, vertcolors)
DOVPTL (COLMOD, PNTCNT, VTXLOC, VTXNRM, VTXCLR)
Create a variable point list primitive object

DoVarSimplePolygonMesh (colormodel, vertexcount, vertlocs, vertnorms, vertcolors, polygoncount, contours, vertexlist, shape, smoothflag)
DOVSPM (COLMOD, VTXCNT, VTXLOC, VTXNRM, VTXCLR, PLYCNT, CONTURS, VTXLST, SHAPE, SMOOFL)
Create a variable simple polygon mesh primitive object

DoVarTriangleMesh (colormodel, vertexcount, vertlocs, vertnorms, vertcolors, trianglecount, triangles, smoothflag)
DOVTRM (COLMOD, VTXCNT, VTXLOC, VTXNRM, VTXCLR, TRICNT, TRIS, SMOOFL)
Create a variable triangle mesh primitive object

DoView ()
DOVW ()
Create a view organizational object

DpUpdVarLineList (object, linecount, vertlocs, vertnorms, vertcolors)
DPUVLL (OBJECT, LINCNT, VTXLOC, VTXNRM, VTXCLR)
Update a variable line list primitive object

DpUpdVarPointList (object, linecount, vertlocs, vertnorms, vertcolors)
DPUVPL (OBJECT, LINCNT, VTXLOC, VTXNRM, VTXCLR)
Update a variable point list primitive object

DpUpdVarSimplePolygonMesh (object, vertlocs, vertnorms, vertcolors, shape, decompose, recompute_norms)
DPUVSM (OBJECT, VTXLOC, VTXNRM, VTXCLR, SHAPE, DECOMP, CMPNRM)
Update a variable simple polygon mesh primitive object

DpUpdVarTriangleMesh (object, vertlocs, vertnorms, vertcolors, recompute_norms)
DPUVTM (OBJECT, VTXLOC, VTXNRM, VTXCLR, CMPNRM)
Update a variable triangle mesh primitive object

DsCompBoundingVol (volume, object)
DSCBV (VOLUME, OBJECT)
Compute dimensions of a bounding volume for an object

DsExecuteObj (object)
DSEO (OBJECT)
Execute an object immediately

DsExecutionAbort ()
DSEA ()
Abort traversal of the current method

DsExecutionReturn ()
DSER ()
Abort traversal of the current group

DsFileRasterRead (filename, width, height, depth, type, data)
DSFRSR (FILENM, FLEN, WIDTH, HEIGHT, DEPTH, TYPE, DATA)
Read raster information from a file

DsHoldObj (object)
DSHO (OBEJCT)
Place a hold on an object

DsInitializeSystem (processors)
DSINIT (PROS)
Initialize Doré

DsInputValue (slot, value)
DSIV (SLOT, VALUE)
Post a value to an input slot

DsInqClassId (class_name)
DSQCI (CLSNME, N)
Return the class identifier of a named class

DsInqCurrentMethod ()
DSQCM ()
Return the current method being executed

DsInqErrorMessage (errornumber, bufbytes, buf, severity)
DSQEM (ERRNUM, BUFSIZ, BUF, SEVERT)
Return an error message

DsInqErrorVars (errorfile, errorhandler)
DSQEV (ERRFIL, ERRHND)
Return current error file and error handler

DsInqExeDepthLimit ()
DSQEDL ()
Return the maximum depth to which objects will be executed

DsInqHoldObj (object)
DSQHO (OBJECT)
Return whether a hold has been placed on the object

DsInqMethodId (method_name)
DSQMI (METNME, N)
Return the method identifier of a named method

DsInqNumRenderers ()
DSQNR ()
Return the number of installed renderers

DsInqObj (object_name_type, object_name, object_type)
DSQOI (OBJNUM, OBJTYP)
DSQOS (OBJSTR, N, OBJTYP)
Return an object specified by name

DsInqObjName (object, object_name_type, object_name)
DSQONT (OBJECT)
DSQONI (OBJECT)
DSQONS (OBJECT, FNAME, LENGTH)
Return an object's name from its object handle

DsInqObjStatus (object)
DSQVOS (OBJECT)
Check the existence of an object

DsInqObjType (object)
DSQOT (OBJECT)
Return an object's type from its object handle

DsInqRendererId (renderer_name)
DSQRI (RENNME, N)
Return the renderer identifier of a named renderer

DsInqRendererNames (names)
DSQRNS (NAMES, LENGTH)
Return a list of the names of the installed renderers

DsInqSafeFlag ()
DSQSF ()
Query the safe flag

DsInqValuatorGroup (slot)
DSQVG (SLOT)
Return the handle for an input slot's valuator group

DsInqVersion (version)
DSQVER (VERSN, LENGTH)
Return the string describing the current version

DsPrintObj (object)
DSPO (OBJECT)
Print information about an object

DsRasterUpdate (raster)
DSRSU (RASTER)
Update a raster object

DsRasterWrite (raster, filename)
DSRSW (RASTER, FNAME, FLEN)
Write a raster object to a file

DsReleaseObj (object)
DSRO (OBJECT)
Release a hold previously placed on an object

DsSetErrorVars (errorfile, errorhandler)
DSSEV (ERRFIL, ERRHND)
Specify an error file and error handler

DsSetExeDepthLimit (limit)
DSSED (LIMIT)
Specify the maximum allowed depth to which objects will be executed

DsSetObjName (object, name_type, object_name, replace)
DSSOND (OBJECT)
DSSONI (OBJECT, OBJNUM, REPL)
DSSONS (OBJECT, OBJSTR, N, REPL)
Set the name of an object

DsSetSafeFlag (flag)
DSSSF (FLAG)
Set the safe flag

DsTerminateSystem ()
DSTERM ()
Terminate Doré

DsTextureUVCount (count)
DSTUVC (COUNT)
Set the number of **uv** sets in the vertex type specification

DsTextureUVWCount (count)
DSTWC (COUNT)
Set the number of **uvw** coordinates in the vertex type specification

DsUpdateAllViews ()
DSUAV ()
Update all views

DsValuatorSwitch (switchvalue)
DSVS (SWVAL)
Enable or disable valuators

DvInqActiveCamera (view)
DVQAC (VIEW)
Return the active camera for a view

DvInqBackgroundColor (view, colormodel, color)
DVQBC (VIEW, COLMOD, COLOR)
Return the background color of a view

DvInqBoundary (view, boundary)
DVQB (VIEW, BNDRY)
Return the view boundary

DvInqClearFlag (view)
DVQCF (VIEW)
Return the clear flag of a view

DvInqDefinitionGroup (view)
DVQDG (VIEW)
Return the definition group for a view

DvInqDisplayGroup (view)
DVQIG (VIEW)
Return the handle for a view's display group

DvInqRendStyle (view)
DVQRS (VIEW)
Return the rendering style of a view

DvInqShadeIndex (view)
DVQSI (VIEW)
Return the shade index of a view

DvInqUpdateType (view)
DVQUT (VIEW)
Return the updatetype of a view

DvSetActiveCamera (view, camera)
DVSAC (VIEW, CAMERA)
Set the active camera for a view

DvSetBackgroundColor (view, colormodel, color)
DVSBC (VIEW, COLMOD, COLOR)
Set the background color of a view

DvSetBoundary (view, boundary)
DVSB (VIEW, BNDRY)
Set the view boundary

DvSetClearFlag (view, clearflag)
DVSCF (VIEW, CLRFLG)
Set the clear flag of a view

DvSetRendStyle (view, renderstyle)
DVSRS (VIEW, RNDSTL)
Set the rendering style of a view

DvSetShadeIndex (view, index)
DVSSI (VIEW, INDEX)
Set the shade index of a view

DvSetUpdateType (view, updatetype)
DVSUT (VIEW, UPDTYP)
Set the updatetype of a view

DvUpdate (view)
DVU (VIEW)
Redisplay the specified view

Fortran Cross Reference

DDP	DdPick
DDPO	DdPickObjs
DDQCE	DdInqColorEntries
DDQCTS	DdInqColorTableSize
DDQE	DdInqExtent
DDQFR	DdInqFrame
DDQFT	DdInqFonts
DDQNF	DdInqNumFonts
DDQPA	DdInqPickAperture
DDQPC	DdInqPickCallback
DDQPPO	DdInqPickPathOrder
DDQPXD	DcInqPixelData
DDQR	DdInqResolution
DDQSM	DdInqShadeMode
DDQSR	DdInqShadeRanges
DDQV	DdInqViewport
DDQVT	DdInqVisualType
DDSCE	DdSetColorEntries
DDSDV	DdSetViewPort
DDSF	DdSetFrame
DDSPA	DdSetPickAperture
DDSPCB	DdSetPickCallback
DDSPPO	DdSetPickPathOrder
DDSSM	DdSetShadeMode
DDSSR	DdSetShadeRanges
DDU	DdUpdate
DEAC	DeAddClass
DEAMTH	(Fortran Only)
DECO	DeCreateObject
DEDO	DeDeleteObject
DEDOD	(Fortran Only)

DEEA	DeExecuteAlternate
DEIOP	DeInitializeObjPick
DEQP	DeInqPickable
DEQR	DeInqRenderable
DEROD	(Fortran Only)
DEWOD	(Fortran Only)
DFQB	DfInqBoundary
DFQJ	DfInqJust
DFQVG	DfInqViewGroup
DFSB	DfSetBoundary
DFSJ	DfSetJust
DFU	DfUpdate
DGAO	DgAddObj
DGAOG	DgAddObjToGroup
DGCK	DgCheck
DGCS	DgClose
DGDE	DgDelEle
DGDEL	DgDelEleBetweenLabels
DGDER	DgDelEleRange
DGE	DgEmpty
DGO	DgOpen
DGQEP	DgInqElePtr
DGQO	DgInqOpen
DGQOP	DgInqObjAtPos
DGQS	DgInqSize
DGRO	DgReplaceObj
DGROG	DgReplaceObjInGroup
DGSEP	DgSetElePtr
DGSEPL	DgSetElePtrRelLabel
DOAMBI	DoAmbientIntens
DOAMBS	DoAmbientSwitch
DOANNT	DoAnnoText
DOBFC	DoBackfaceCullable
DOBFCS	DoBackfaceCullSwitch
DOBV	DoBoundingVol
DOBVS	DoBoundingVolSwitch
DOCB	DoCallback
DOCM	DoCamera
DOCMX	DoCameraMatrix
DOCS	DoClipSwitch
DOCV	DoClipVol
DOD	DoDevice
DODC	DoDepthCue
DODCS	DoDepthCueSwitch
DODIFC	DoDiffuseColor
DODIFI	DoDiffuseIntens

DODIFS	DoDiffuseSwitch
DODP	DoDataPtr
DODV	DoDataVal
DOES	DoExecSet
DOFL	DoFilter
DOFR	DoFrame
DOFRS	DoFileRaster
DOG	DoGroup
DOGRMO	DoGlbRendMaxObjs
DOGRMS	DoGlbRendMaxSub
DOGRRL	DoGlbRendRayLevel
DOGTUV	DoGenerateTextureUV
DOHSS	DoHiddenSurfSwitch
DOILG	DoInLineGroup
DOINVS	DoInvisSwitch
DOIS	DoInputSlot
DOIT	DoInterpType
DOLAF	DoLookAtFrom
DOLC	DoLightColor
DOLI	DoLightIntens
DOLINL	DoLineList
DOLL	DoLabel
DOLNT	DoLineType
DOLT	DoLight
DOLTA	DoLightAttenuation
DOLTS	DoLightSwitch
DOLTSA	DoLightSpreadAngles
DOLTSE	DoLightSpreadExp
DOLTT	DoLightType
DOLW	DoLineWidth
DOM	DoMatrix
DOMBVE	DoMinBoundingVolExt
DOMF	DoMarkerFont
DOMG	DoMarkerGlyph
DOMS	DoMarkerScale
DONRBS	DoNURBSurf
DONS	DoNameSet
DOPAR	DoParallel
DOPAT	DoPatch
DOPER	DoPerspective
DOPGN	DoPolygon
DOPGNM	DoPolygonMesh
DOPID	DoPickID
DOPL	DoPolyline
DOPM	DoPolymarker
DOPMS	DoPrimSurf

DOPNTL	DoPointList
DOPPA	DoPopAtts
DOPPMX	DoPopMatrix
DOPRJ	DoProjection
DOPS	DoPickSwitch
DOPUA	DoPushAtts
DOPUMX	DoPushMatrix
DOREFS	DoReflectionSwitch
DOREPT	DoRepType
DORFRI	DoRefractionIndex
DORFRS	DoRefractionSwitch
DOROT	DoRotate
DORS	DoRaster
DOSADP	DoSampleAdaptive
DOSASW	DoSampleAdaptiveSwitch
DOSC	DoScale
DOSDS	DoSubDivSpec
DOSFLT	DoSampleFilter
DOSHAS	DoShadowSwitch
DOSHR	DoShear
DOSI	DoShadeIndex
DOSJIT	DoSampleJitter
DOSJSW	DoSampleJitterSwitch
DOSPCC	DoSpecularColor
DOSPCF	DoSpecularFactor
DOSPCI	DoSpecularIntens
DOSPCS	DoSpecularSwitch
DOSPGN	DoSimplePolygon
DOSPM	DoSimplePolygonMesh
DOSRFS	DoSurfaceShade
DOSSPR	DoSampleSuper
DOSSSW	DoSampleSuperSwitch
DOSTER	DoStereo
DOSTES	DoStereoSwitch
DOTA	DoTextAlign
DOTAA	DoTextureAntiAlias
DOTC	DoTranspColor
DOTEF	DoTextExpFactor
DOTF	DoTextFont
DOTH	DoTextHeight
DOTI	DoTranspIntens
DOTMB	DoTextureMapBump
DOTMBS	DoTextureMapBumpSwitch
DOTMDC	DoTextureMapDiffuseColor
DOTMDS	DoTextureMapDiffuseColorSwitch
DOTME	DoTextureMapEnviron

DOTMES	DoTextureMapEnvironSwitch
DOTMTI	DoTextureMapTranspIntens
DOTMTS	DoTextureMapTranspIntensSwitch
DOTMX	DoTransformMatrix
DOTOC	DoTranspOrientColor
DOTOE	DoTranspOrientExp
DOTOI	DoTranspOrientIntens
DOTOP	DoTextureOp
DOTOR	DoTorus
DOTOS	DoTranspOrientSwitch
DOTPA	DoTextPath
DOTPR	DoTextPrecision
DOTRIL	DoTriangleList
DOTRIM	DoTriangleMesh
DOTS	DoTranspSwitch
DOTSP	DoTextSpace
DOTSUV	DoTextureScaleUV
DOTSW	DoTextureScaleUVW
DOTTUV	DoTextureTranslateUV
DOTTW	DoTextureTranslateUVW
DOTUV	DoTextUpVector
DOTUVI	DoTextureUVIndex
DOTWI	DoTextureUVWIndex
DOTXT	DoText
DOTXUV	DoTextureExtendUV
DOTXW	DoTextureExtendUVW
DOVLNL	DoVarLineList
DOVPTL	DoVarPointList
DOVSPM	DoVarSimplePolygonMesh
DOVTRM	DoVarTriangleMesh
DOVW	DoVarPointList
DOXLT	DoTranslate
DPUVLL	DpUpdateVarLineList
DPUVPL	DpUpdateVarPointList
DPUVSM	DpUpdateVarSimplePolygonMesh
DPUVTM	DpUpdateVarTriangleMesh
DSCBV	DsCompBoundingVol
DSEA	DsExecutionAbort
DSEO	DsExecuteObj
DSER	DsExecutionReturn
DSFRSR	DsFileRasterRead
DSHO	DsHoldObj
DSINIT	DsInitializeSystem
DSIV	DsInputValue
DSPO	DsPrintObj
DSQCI	DsInqClassId

DSQCM	DsInqCurrentMethod
DSQEDL	DsInqExeDepthLimit
DSQEM	DsInqErrorMessage
DSQEV	DsInqErrorVars
DSQHO	DsInqHoldObj
DSQMI	DsInqMethodId
DSQNR	DsInqNumRenderers
DSQOI	DsInqObj
DSQONI	DsInqObjName
DSQONS	DsInqObjName
DSQONT	DsInqObjName
DSQOS	DsInqObj
DSQOT	DsInqObjType
DSQRI	DsInqRendererId
DSQRNS	DsInqRendererNames
DSQSF	DsInqSafeFlag
DSQVER	DsInqVersion
DSQVG	DsInqValuatorGroup
DSQVOS	DsInqObjStatus
DSRO	DsReleaseObj
DSRSU	DsRasterUpdate
DSRSW	DsRasterWrite
DSSEDL	DsSetExeDepthLimit
DSSEV	DsSetErrorVars
DSSOND	DsSetObjName
DSSONI	DsSetObjName
DSSONS	DsSetObjName
DSSSF	DsSetSafeFlag
DSTERM	DsTerminateSystem
DSTUVC	DsTextureUVCount
DSTWC	DsTextureUVWCount
DSUAV	DsUpdateAllViews
DSVS	DsValuatorSwitch
DVQAC	DvInqActiveCamera
DVQB	DvInqBoundary
DVQBC	DvInqBackgroundColor
DVQCF	DvInqClearFlag
DVQDG	DvInqDefinitionGroup
DVQIG	DvInqDisplayGroup
DVQRS	DvInqRendSyle
DVQSI	DvInqShadeIndex
DVQUT	DvInqUpdateType
DVSAC	DvSetActiveCamera
DVSB	DvSetBoundary
DVSBC	DvSetBackgroundColor
DVSCF	DvSetClearFlag

DVSRS	DvSetRendStyle
DVSSI	DvSetShadeIndex
DVSUT	DvSetUpdateType
DVU	DvUpdate

DORÉ MATRIX TRANSFORMATIONS

CHAPTER FIVE

Matrix Format

This chapter documents the specific matrix forms that are used by the Doré system. Doré uses column vectors to represent points so the general form to transform a point is:

$$\begin{bmatrix} p_x' \\ p_y' \\ p_z' \\ p_w' \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \times \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix}$$

Pre-concatenating matrix $\overline{A}$ to matrix $\overline{B}$ means that the transformation $\overline{A}$ is applied to the points before the transformation $\overline{B}$. Remember that when using column vectors pre-concatenation implies post-multiplication and post-concatenation implies pre-multiplication. This means that pre-concatenating matrix $\overline{A}$ to matrix $\overline{B}$ results in the matrix multiplication $\overline{B} \times \overline{A}$.

Transformation Pipeline

The complete transformation pipeline is:

$$\begin{bmatrix} p_x' \\ p_y' \\ p_z' \\ p_w' \end{bmatrix} = \overline{V} \times \overline{C} \times \overline{CTM} \times \begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix}$$

where:

$\overline{V}$ - viewing transform
$\overline{C}$ - camera positioning transform
$\overline{CTM}$ - current transformation matrix
which is the composite modeling transform

The camera positioning transform $\overline{C}$ is determined during rendering initialization, and is the current modeling transformation at the time the *DoCamera* object is executed. It can be formed by any combination of modeling tranformation objects, but is usually done with the *DoLookAtFrom* command.

Modeling Transforms

The following commands create objects in the Doré system that modify the modeling coordinate system. This coordinate system is a right handed coordinate system in units convenient to the user. When the objects are executed during traversal they cause their matrix to be pre-concatenated to the current transformation matrix. For instance, if an object generates a matrix $\overline{X}$ then when the object is executed during traversal the matrix $\overline{X}$ will be pre-concatenated to the current transformation matrix $\overline{CTM}$ to create the new current transformation matrix $\overline{CTM}'$. In symbols this is $\overline{CTM}' = \overline{CTM} \times \overline{X}$.

DoTranslate (t_x, t_y, t_z) generates the matrix $\overline{T}$, where:

$$\overline{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DoScale (s_x, s_y, s_z) generates the matrix $\overline{S}$, where:

$$\overline{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DoRotate (*axis*, θ) generates the matrix $\overline{R}$, where:

$$\overline{R} = \begin{cases} \overline{R_x} & \text{if } axis \equiv \text{DcXAxis} \\ \overline{R_y} & \text{if } axis \equiv \text{DcYAxis} \\ \overline{R_z} & \text{if } axis \equiv \text{DcZAxis} \end{cases}$$

and

$$\overline{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\overline{R_y} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\overline{R_z} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DoShear (*plane*, d_1, d_2) generates the matrix $\overline{Sh}$, where:

$$\overline{Sh} = \begin{cases} \overline{Sh_{yz}} & \text{if } plane \equiv \text{DcYZ} \\ \overline{Sh_{xz}} & \text{if } plane \equiv \text{DcXZ} \\ \overline{Sh_{xy}} & \text{if } plane \equiv \text{DcXY} \end{cases}$$

and

$$\overline{Sh_{yz}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ d_1 & 1 & 0 & 0 \\ d_2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\overline{Sh_{xz}} = \begin{bmatrix} 1 & d_1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & d_2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\overline{Sh_{xy}} = \begin{bmatrix} 1 & 0 & d_1 & 0 \\ 0 & 1 & d_2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The Doré command *DoTransformMatrix* ($\overline{X}$, *comptype*) creates an object that modifies the current transformation matrix $\overline{CTM}$ when it is executed as follows:

$$\overline{CTM}' = \begin{cases} \overline{X} & \text{if } comptype \equiv \text{DcReplace} \\ \overline{CTM} \times \overline{X} & \text{if } comptype \equiv \text{DcPreConcatenate} \\ \overline{X} \times \overline{CTM} & \text{if } comptype \equiv \text{DcPostConcatenate} \end{cases}$$

Note that this is the only object that allows the user to replace the current transformation matrix or to post-concatenate another matrix to the current transformation matrix. All other commands only pre-concatenate a matrix to the current transformation matrix.

It is important to note that because Doré uses column vectors, and pre-concatenation means placing the matrix closest to the points, the matrix $\overline{X}$ will be post-multiplied with the composite matrix $\overline{CTM}$. Similarly with post-concatenation, the matrix $\overline{X}$ will be pre-multiplied with the composite matrix $\overline{CTM}$.

Camera and Light Transform

The *DoLookAtFrom* ($\overline{at}$, $\overline{from}$, $\overline{up}$) generates the matrix $\overline{L}$ that is pre-concatenated to the current transformation matrix.

$$\overline{L} = \begin{bmatrix} u_x & v_x & n_x & from_x \\ u_y & v_y & n_y & from_y \\ u_z & v_z & n_z & from_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

$$\overline{n} = \overline{from} - \overline{at}$$
$$\overline{u} = \overline{up} \times \overline{n}$$
$$\overline{v} = \overline{n} \times \overline{u}$$

The *DoLookAtFrom* object generates a matrix that transforms the current coordinate system so that it forms a right-handed coordinate system with $\overline{from}$ at the origin, $\overline{up}$ aligned to the positive y axis and $\overline{from} - \overline{at}$ aligned to the positive z axis. This command is especially useful for positioning lights and cameras.

Viewing Transformations

The following matrices are used to define the volume of world space that will be visible in the final image. The frustrum space coordinate system is a right handed system with Y up, X to the right and Z coming out of the screen. The eye is positioned at the origin looking down the negative Z axis. The frustrum extent, used for clipping, is defined as:

$$-w \le x \le w$$
$$-w \le y \le w$$
$$-w \le z \le 0$$

For both the perspective and parallel projection transformations, the matrices are pre-concatenated with a scaling matrix $\overline{S_v}$ to handle the aspect ratio of the view, where:

$$\overline{S_v} = \begin{cases} \overline{S_{y/x}} & \text{if } viewsize_x > viewsize_y \\ \overline{S_{x/y}} & \text{if } viewsize_x \leq viewsize_y \end{cases}$$

and

$$\overline{S_{y/x}} = \begin{bmatrix} \frac{viewsize_y}{viewsize_x} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\overline{S_{x/y}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{viewsize_x}{viewsize_y} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$viewsize_x = fur_x - bll_x$$
$$viewsize_y = fur_y - bll_y$$

where $\overline{fur}$ and $\overline{bll}$ are fields of the view boundary returned by *DvInqBoundary*.

DoPerspective (*fov, hither, yon*) creates an object that generates the camera projection matrix $\overline{V}$, where:

$$\overline{V} = \begin{bmatrix} \frac{1}{\tan\left[\frac{fov}{2}\right]} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left[\frac{fov}{2}\right]} & 0 & 0 \\ 0 & 0 & \frac{1}{1-\frac{hither}{yon}} & \frac{-hither}{1-\frac{hither}{yon}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \overline{S_v}$$

DoParallel (*winsize, hither, yon*) creates an object that generates the camera projection matrix $\overline{V}$, where:

$$\overline{V} = \begin{bmatrix} \frac{2}{winsize} & 0 & 0 & 0 \\ 0 & \frac{2}{winsize} & 0 & 0 \\ 0 & 0 & \frac{1}{hither - yon} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -hither \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \overline{S_v}$$

DoProjection (*window, type, prp, plane, hither, yon*) creates an object that generates the camera projection matrix $\overline{V}$. Note that both *hither* and *yon* must be negative.

$$\overline{V} = \begin{cases} \overline{V_{perspective}} \times \overline{M} & \text{if } type \equiv \text{DcPerspective} \\ \overline{V_{parallel}} \times \overline{M} & \text{if } type \equiv \text{DcParallel} \end{cases}$$

where:

$$\overline{M} = \begin{bmatrix} 1 & 0 & \frac{\left[-\frac{(bll_x + fur_x)}{2} - prp_x\right]}{vp} & 0 \\ 0 & 1 & \frac{\left[-\frac{(bll_y + fur_y)}{2} - prp_y\right]}{vp} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -prp_x \\ 0 & 1 & 0 & -prp_y \\ 0 & 0 & 1 & -prp_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\overline{V_{perspective}} = \begin{bmatrix} \frac{vp}{xfur_x} & 0 & 0 & 0 \\ 0 & \frac{vp}{xfur_y} & 0 & 0 \\ 0 & 0 & \frac{1}{1-\frac{h}{y}} & \frac{-h}{1-\frac{h}{y}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\overline{V_{parallel}} = \begin{bmatrix} \frac{1}{xfur_x} & 0 & 0 & 0 \\ 0 & \frac{1}{xfur_y} & 0 & 0 \\ 0 & 0 & \frac{1}{h-y} & \frac{-h}{h-y} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$vp = prp_z - plane$$
$$h = hither - prp_z$$
$$y = yon - prp_z$$

$$\begin{bmatrix} xfur_x \\ xfur_y \\ xfur_z \end{bmatrix} = \overline{M} \times \begin{bmatrix} fur_x \\ fur_y \\ plane \end{bmatrix}$$

NOTE: $\overline{bll}$ and $\overline{fur}$ are fields of *window*.

DoCameraMatrix ($\overline{V}$) creates an object that sets the camera projection matrix to the user defined matrix $\overline{V}$ when it is executed.

Program Examples

The following two examples show how additional types of transforms could be created using the existing Doré transform objects.

Example 1 - RotateAxis

The first example is a complete C function/Fortran subroutine that pre-concatenates a matrix that does a rotation about an arbitrary axis passing through the current origin.

RotateAxis (x,y,z,θ) generates the matrix $\overline{R_{axis}}$ where:

$$\overline{R_{axis}} = \begin{bmatrix} x^2 + (1-x^2)\cos(\theta) & xy(1-\cos(\theta)) - z\sin(\theta) & xz(1-\cos(\theta)) + y\sin(\theta) & 0 \\ xy(1-\cos(\theta)) + z\sin(\theta) & y^2 + (1-y^2)\cos(\theta) & yz(1-\cos(\theta)) - x\sin(\theta) & 0 \\ xz(1-\cos(\theta)) - y\sin(\theta) & yz(1-\cos(\theta)) + x\sin(\theta) & z^2 + (1-z^2)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The C code is as follows:

```
#include <dore.h>
#include <math.h>

rotate_axis (x, y, z, theta)
    DtReal x, y, z;
    DtReal theta;
{
   DtReal costheta;
   DtReal sintheta;
   DtMatrix4x4 matrix;

   costheta = cos (theta);
   sintheta = sin (theta);

   matrix[0][0] = x*x + (1 - x*x)*costheta;
   matrix[0][1] = x*y*(1 - costheta) - z*sintheta;
   matrix[0][2] = x*z*(1 - costheta) + y*sintheta;
   matrix[0][3] = 0.0;

   matrix[1][0] = x*y*(1 - costheta) + z*sintheta;
   matrix[1][1] = y*y + (1 - y*y)*costheta;
   matrix[1][2] = y*z*(1 - costheta) - x*sintheta;
   matrix[1][3] = 0.0;

   matrix[2][0] = x*z*(1 - costheta) - y*sintheta;
   matrix[2][1] = y*z*(1 - costheta) + x*sintheta;
   matrix[2][2] = z*z + (1 - z*z)*costheta;
   matrix[2][3] = 0.0;
```

```
    matrix[3][0] = 0.0;
    matrix[3][1] = 0.0;
    matrix[3][2] = 0.0;
    matrix[3][3] = 1.0;

    DoTransformMatrix (matrix, DcPreConcatenate);
}
```

The Fortran code for the same operation is:

```
C
      SUBROUTINE ROTAX(X,Y,Z,THETA)
C
C
      INCLUDE '/usr/include/fortran/DORE'
      INCLUDE '/usr/include/fortran/DORETYPES'
C
      INTEGER*4 NEWMAT
      REAL*8 CTHETA
      REAL*8 STHETA
      REAL*8 MATRIX(4,4)
C
      CTHETA = COS(THETA)
      STHETA = SIN(THETA)
C
      MATRIX(1,1) = X*X+(1-X*X)*CTHETA
      MATRIX(2,1) = X*Y*(1-CTHETA)-Z*STHETA
      MATRIX(3,1) = X*Z*(1-CTHETA)+Y*STHETA
      MATRIX(4,1) = 0.0
C
      MATRIX(1,2) = X*Y*(1-CTHETA)+Z*STHETA
      MATRIX(2,2) = Y*Y+(1-Y*Y)*CTHETA
      MATRIX(3,2) = Y*Z*(1-CTHETA)-X*STHETA
      MATRIX(4,2) = 0.0
C
      MATRIX(1,3) = X*Z*(1-CTHETA)-Y*STHETA
      MATRIX(2,3) = Y*Z*(1-CTHETA)+X*STHETA
      MATRIX(3,3) = Z*Z+(1-Z*Z)*CTHETA
      MATRIX(4,3) = 0.0
C
      MATRIX(1,4) = 0.0
      MATRIX(2,4) = 0.0
      MATRIX(3,4) = 0.0
      MATRIX(4,4) = 1.0
C
      NEWMAT = DOTMX (MATRIX, DCPREC)
C
      RETURN
      END
```

Example 2 - RotateCenter

The second example shows a combination of Doré calls can be combined to form a composite matrix that rotates about the x, y, and z axes centered at a point $\vec{c}$.

$$RotateCenter\ (\vec{c}, r_x, r_y, r_z) = T_{\vec{c}}\, R_z\, R_y\, R_x\, T_{-\vec{c}}$$

This matrix is generated in Doré by the following sequence in C:

```
DoTranslate (cx, cy, cz);
DoRotate (DcZAxis, rz);
DoRotate (DcYAxis, ry);
DoRotate (DcXAxis, rx);
DoTranslate (-cx, -cy, -cz);
```

The identical Fortran sequence is:

```
ITRANS1 = DOXLT (CX, CY, CZ)
IROT1 = DOROT (DCZAX, RZ)
IROT2 = DOROT (DCYAX, RY)
IROT3 = DOROT (DCXAX, RX)
ITRANS2 = DOXLT (-CX, -CY, -CZ)
```

DORÉ NAMING CONVENTIONS

APPENDIX A

All Doré functions start with a D followed by a single lower case letter specifying the type of Doré function. The rest of the function name is a mnemonic, using concatenated words or abbreviations with the first character of each word in upper case.

NOTE
In Fortran the entire function name is in upper case.

DcXXXXX	a Doré *constant* value, used as a parameter
DdXXXXX	a Doré *device* function
DeXXXXX	a Doré user *extension* function Used when adding user-defined extensions to Doré; not typically used by application programs.
DfXXXXX	a Doré *frame* function
DgXXXXX	a Doré *group* function
DoXXXXX	a Doré *object creation* function
DpXXXXX	a Doré *primitive* function
DsXXXXX	a Doré *system*, or object manipulation function
DtXXXXX	a Doré *type* declaration, used for passing parameters
DvXXXXX	a Doré *view* function

DORÉ VERTEX TYPES

APPENDIX B

The information in this appendix pertains to most Doré primitive objects. Notable exceptions are variable data primitives (*DoVar-<DOV>* functions), which use a different syntax to obtain vertex information, and *DoPrimSurf <DOPMS>* primitives, which do not allow user-defined vertex information.

When entering primitive object data into the Doré database, you may provide information about vertex normals, vertex colors, *uv* coordinates and *uvw* coordinates. This additional information is used by Doré to more accurately shade and texture the object during rendering.

The Vertex Type Parameter

When a Doré primitive object is created, all the vertex data (locations, normals, colors, *uv* coordinates, and *uvw* coordinates) are specified in a single array. A base *vertextype* parameter must be set to reflect the type of information supplied for each vertex. The following table lists the possible base values for *vertextype*:

Value of *vertextype*		Vertex Information Supplied
DcLoc	*DCL*	Only the vertex location
DcLocNrm	*DCLN*	The vertex location and the vertex normal
DcLocClr	*DCLC*	The vertex location and the vertex color
DcLocNrmClr	*DCLNC*	All three are supplied

NOTE
Some colormodels require more or less than three components per vertex.

If *DcLoc <DCL>* is specified, three floating point values (x,y,z) must be supplied for each vertex. *DcLocNrm <DCLN>* and *DcLocClr <DCLC>* require three additional values for vertex normals and vertex colors, respectively. *DcLocNrmClr <DCLNC>* requires nine floating point values for each vertex.

Vertex normals *must* be unit vectors (of length one). Normals that are not unit vectors will produce improper shading.

Vertex Types that Contain Texture Information

The Doré functions *DsTextureUVCount <DSTUVC>* and *DsTextureUVWCount <DSTWC>* can be used in conjunction with a base *vertextype* parameter to specify a new vertex type that includes *uv* and/or *uvw* coordinates.

The new vertex type is specifed by ORing one of the base vertex values with the above named functions. Both *DsTextureUVCount <DSTUVC>* and *DsTextureUVWCount <DSTWC>* take a parameter, *count*, which specifies how many sets of *uv* or *uvw* coordinates are specified for each vertex in the vertex list.

As an example, consider the following vertex type specification within a *DoTriangleList* function:

DoTriangleList(DcRGB, DcLocClr | DsTextureUVCount(2), tri_cnt, vert)

In this function, the vertex type specifies that each vertex is defined by ten DtReal values:

an ordered triple specifying the vertex position
an ordered triple specifying the red, green and blue values
two ordered pairs specifying *uv* coordinates

If the value of *tri_cnt* in this example were fifteen, the array *vert* would contain $10 \times 15 = 150$ DtReal values.

A vertex type may specify both *uv* and *uvw* coordinates. Associated vertex arrays must list *uv* data before *uvw* data.

Other examples of vertex arrays can be found in Chapter Five of the *Doré Programmer's Guide*.

RASTER FILE AND MEMORY FORMATS

APPENDIX C

Doré allows the representation and storage of two- and three-dimensional image data in raster objects. Doré has two reperesentations for raster data, a file representation and a memory representation. This appendix describes the exact formats used by Doré for communicating raster data.

Raster Pixel and Voxel Component Semantics

A *pixel* (picture element) is the set of information associated with a specific point in a two-dimensional raster image. A *voxel* (volume element) is the three-dimensional analog of a pixel, and is used when referring to three-dimensional raster objects. In Doré, a pixel or voxel consists of certain combinations of RGB, alpha, and Z image data. This section describes the meaning of the values of these pixel and voxel components.

RGB Data. RGB color information is always stored together, and is represented by three 8-bit unsigned integer byte values. Each value represents the intensity of light for that color channel. A value of 0 means no light, and a value of 255 means full intensity light.

Alpha Data. Alpha information describes the transparency of the pixel or voxel, and is represented as a single 8-bit unsigned-integer byte value. A value of 0 means the pixel or voxel is opaque, and a value of 255 means it is completely transparent.

Z Data. The Z information describes the relative distance of the pixel or voxel, and is stored as a 32-bit unsigned integer. A value of 0 represents a distance as far away as possible, and a value of 4,294,967,296 represents a distance as close as possible.

Doré Raster File Format

Raster files are used to store Doré image data in computer file systems and to allow other programs to communicate image data to Doré.

Doré reads and writes raster files compatible with the Doré raster file format described here. Raster files of this format can be read and written by Doré programs with the *DsFileRasterRead* and *DsRasterWrite* calls, respectively. The Rasterfile device driver also writes image files using this format.

Doré raster files consist of an ASCII header section followed by a binary data section.

An example of a raster file (with the binary data shown only as an annotation) is given below:

```
# brick texture raster - Dore' Rasterfile
        rastertype = image
        width = 128
        height = 128
        depth = 1
        pixel = r8g8b8z32
        wordbyteorder = big-endian
<ff><ff>
        .
        .
    (binary data)
        .
        .
<EOF>
```

The sections below describe the formats of the header and binary data sections of raster files in more detail.

Raster File Header Section

The ASCII header of a Doré raster file consists of a several attribute-value pairs followed by an end-of-header marker.

An attribute-value pair is an attribute name followed by an "=" symbol, followed by a numeric value or an identifier name value. Optional white space characters (spaces, tabs, newlines/linefeeds, or carriage returns) may precede or follow the "=".

Attribute-value pairs must be separated by whitespace characters. The convention is to have one attribute-value pair per line.

Unknown attribute-value pairs will be ignored. An attribute may have multiple values, separated by commas, even though this feature is not currently used by Doré.

The end-of-header marker is a formfeed character, followed by zero or more non-formfeed characters, followed by a formfeed character. The double formfeed end-of-header marker allows the header section to be expanded to fall at the end of a disk block boundary. This improves efficiency on some machines.

Comments consist of the "#" symbol and any number of characters to the end of the line.

The attributes *rastertype, width, height,* and *pixel* are mandatory. The *rastertype* attribute is required to be the first attribute in the header. The order of all attributes other than *rastertype* is not restricted. Attribute names and value strings are case sensitive.

Mandatory and optional attributes are listed below with their permissible values.

rastertype
(*mandatory; must be first attribute*) must have the value string *image*; it indicates that this is a Doré raster image file.

width
(*mandatory*) takes an unsigned integer value indicating the width of the raster image in pixels (or voxels).

height
(*mandatory*) takes an unsigned integer value indicating the height of the raster image in pixels (or voxels).

depth
(*optional - defaults to 1*) takes an unsigned integer value indicating the depth of the raster image in pixels (voxels).

wordbyteorder
(*optional - defaults to big-endian*) indicates the byte order of large unsigned integer binary numbers stored in the file (used for Z values). It takes one of two string values:

big-endian
meaning the machine represents words in the order of most significant byte to least significant byte.

little-endian

meaning the machine represents words in the order of least significant byte to most significant byte.

pixel

(*mandatory*) indicates the content of each pixel in the raster image and takes one of the following string values:

r8g8b8

indicates that each pixel in the image consists of three 8-bit values, representing the RGB color components of the pixel. This is equivalent to the Doré *DtRasterType* value *DcRasterRGB* <*DCRRGB*>.

r8g8b8a8

indicates that each pixel in the image consists of four 8-bit values, representing the RGB color components and the transparency component of the pixel. This is equivalent to the Doré *DtRasterType* value *DcRasterRGBA* <*DCRRA*>.

r8g8b8a8z32

indicates that each pixel in the image consists of four 8-bit values and one 32-bit value, representing the RGB color components, the transparency component, and the Z depth component of the pixel. This is equivalent to the Doré *DtRasterType* value *DcRasterRGBAZ* <*DCRRAZ*>.

r8g8b8z32

indicates that each pixel in the image consists of three 8-bit values and one 32-bit value, representing the RGB color components, and the Z depth component of the pixel. This is equivalent to the Doré *DtRasterType* value *DcRasterRGBZ* <*DCRRZ*>.

a8

indicates that each pixel in the image consists of one 8-bit value representing the transparency component of the pixel. This is equivalent to the Doré *DtRasterType* value *DcRasterA* <*DCRA*>.

z32

indicates that each pixel in the image consists of one 32-bit value representing the Z depth component of the pixel. This is equivalent to the Doré *DtRasterType* value *DcRasterZ* <*DCRZ*>.

Below is shown an example of the minimal number of header attributes acceptable.

```
# mountain background raster - Dore' Rasterfile
        rastertype = image
        width = 1280
        height = 1024
        pixel = r8g8b8r8g8b8
<ff><ff>
```

Raster File Binary Data Section

Raster image binary data for Doré has six basic formats depending on which of the *r8g8b8, r8g8b8a8, r8g8b8a8z32, r8g8b8z32, a8,* or *z32* types were specified for the pixels or voxels in the raster.

For file based images, the binary image data is logically read from and written to the files in a byte-by-byte fashion. Since RGB and alpha information is represented as 8-bit byte values, their representation and order in the file is the same for all machines. Z information, however, consists of an unsigned 32-bit integer, and while the position of the 32-bit value is the same in the file for all machines, the order of the bytes in the value is machine dependent. If the word byte order on the machine writing the file goes from the most significant byte first down to the least significant byte, the order is called *big-endian*. The ordering may then be optionally indicated in the file header with the *wordbyteorder* = *big-endian* attribute-value pair. If the word byte order on the machine writing the file goes from the least significant byte first up to the most significant byte, the order is called *little-endian*. In that case, the ordering must be indicated in the file header with the *wordbyteorder* = *little-endian* attribute-value pair. Doré can accept raster files written in either fashion.

Binary data in Doré raster files is stored in pixel interleaved fashion, meaning that the data is stored pixel (voxel) by pixel (voxel). The entire contents of each individual pixel (voxel) are stored together.

The order of pixels (voxels) in the binary data section is in width, then height, and then depth order. The data starts with the upper left hand front corner of the image data and scans out in a left to right, top to bottom, front to back order. The index related to *width* will vary the fastest, the index related to *height* will vary the second fastest, and the index related to *depth* will vary the slowest.

The order of the data in a single pixel (voxel) for each of the six formats is:
(each <..> denotes eight bits)

r8g8b8

<red><green><blue>

r8g8b8a8

<red><green><blue><alpha>

r8g8b8a8z32

<red><green><blue><alpha><Z_1><Z_2><Z_3><Z_4>

For big-endian <Z_1> is the most significant byte.
For little-endian <Z_1> is the least significant byte.

r8g8b8z32

<red><green><blue><Z_1><Z_2><Z_3><Z_4>

For big-endian <Z_1> is the most significant byte.
For little-endian <Z_1> is the least significant byte.

a8

<alpha>

z32

<Z_1><Z_2><Z_3><Z_4>

For big-endian <Z_1> is the most significant byte.
For little-endian <Z_1> is the least significant byte.

Raster Memory Format

A memory format for raster data can be used to communicate image information to and from Doré, allowing programs (such as image processing libraries) to manipulate image data supplied to and obtained from Doré.

Doré uses the raster memory format described here. Raster memory data of this format is required when a Doré *DoRaster* object is created. Raster memory data can be obtained from files in the Doré raster file format using the *DsFileRasterRead* call, and from some Doré devices with the *DdInqPixelData* call.

Raster data in memory consists of width, height, depth, type (a *DtRasterType* value) attribute information, and the binary data. It is passed to and obtained from Doré by passing the four attribute values as separate parameters to and from Doré routines along with a pointer to the binary data.

The binary memory format of the raster data is described in the following section.

Raster Memory Binary Data Format

The format of the binary data for raster memory data is identical to the format for the binary file raster data with the exception that word byte order is not a concern to the programmer since the thirty-two bit Z value will be in whatever order is native to the machine.

Creating Raster Files from Raster Memory Data

To create Doré format raster files in various programs, the following code segment is recommended.

```
C code:

DtObject tmp_raster;
DtInt width, height, depth;
DtRasterType type;
DtPtr data;

#include <stdio.h>
#include <dore.h>

                        /* initialize the Dore' system */
DsInitializeSystem(0);
                        /* allocate and initialize binary
```

```
                                image data pointed at by data */
            .
            .
                                /* turn memory data into raster object */
tmp_raster = DoRaster(width,height,depth,type,
                                DcNullPtr,data,DcDeleteData);
                                /* write out the file */
DsRasterWrite(tmp_raster,"myfile");
                                /* clean up */
DsSystemTerminate();


Fortran code:

      INTEGER*4 RSTR, WIDTH, HEIGHT, DEPTH, RTYPE, DATA

      INCLUDE '/usr/include/fortran/DORE'

C ... initialize the Dore' system
      CALL DSINIT(0)
        .
        .
C ... turn memory data into raster object
      RSTR = DORS(WIDTH,HEIGHT,DEPTH,RTYPE,'',0,DATA,DCDELD)
C ... write out the file
      DSRSW(RSTR,'myfile',6)

C ... clean up
      CALL DSTERM
```

This method will automatically take care of any concerns about byte orders when using Z data.

Obtaining Raster Memory Data from Raster Files

To obtain raster data from Doré format raster files, and create a Doré *DoRaster* object, the following code segment is recommended.

```
C code:

DtObject raster;
DtInt width, height, depth;
DtRasterType type;
DtPtr data;

#include <stdio.h>
#include <dore.h>

                /* initialize the Dore' system */
DsInitializeSystem(0);
        .
        .
                /* read the file */
```

```
if (DsFileRasterRead("myfile",&width,&height,
                        &depth,&type,&data)==-1) {
                /* error */
        .
        .
} else {
                /* create Doré raster object */
raster = DoRaster(width,height,depth,type,
                        DcNullPtr,data,DcDeleteData);
}
                /* clean up */
DsSystemTerminate();
```

Fortran code:

```
      INTEGER*4 RSTR
      INTEGER*4 WIDTH, HEIGHT, DEPTH, RTYPE, DATA

      INCLUDE '/usr/include/fortran/DORE'

C ... initialize the Dore' system
      CALL DSINIT(0)
        .
        .
C ... read the file
      IF (DSFRSR('myfile',6,WIDTH,HEIGHT,DEPTH,RTYPE,DATA).EQ.-1) THEN
C ... error
      ELSE
C ... create Doré raster object
         RSTR = DORS(WIDTH,HEIGHT,DEPTH,RTYPE,'',0,DATA,DCDELD)
      ENDIF

C ... clean up
      CALL DSTERM
```

DORÉ RENDERERS

APPENDIX D

Contents of this Appendix

This appendix contains the following subsections:

DORÉ RENDERER OVERVIEW

APPENDIX D1

Renderers and Doré

Doré allows you to choose the renderer used to generate a view. Each renderer supports a particular rendering style. Some renderers produce high quality, lifelike images, while other renderers are faster and produce less realistic images. Because of these differences, some Doré attributes represent functionality that is inappropriate or unimplemented for particular renderers.

This overview lists all the Doré classes and functions and indicates which ones are renderer independent, (that is, have an identical effect with all renderers), and which ones are not. The subsections discuss the particulars of each renderer.

Organizational Classes

All renderers in the Doré system support the organizational object classes listed in Table D1-1.

Table D1-1. Organizational Classes

DoDevice	DoInLineGroup	DoPushAtts
DoFrame	DoPopAtts	DoView
DoGroup		

Miscellaneous Classes

The miscellaneous classes are those classes that cannot be directly rendered, but may be used to represent values in other objects. Table D1-2 lists the miscellaneous object classes.

Table D1-2. Miscellaneous Classes

DoCallback	DoFileRaster	DoMatrix
DoDataPtr	DoInputSlot	DoRaster
DoDataVal	DoLabel	

Studio Classes

The studio object classes listed in Table D1-3 are supported by all renderers.

Table D1-3. Studio Classes

DoCamera	DoLight

Studio Attribute Classes

The studio attribute classes are classes of objects that affect the global environment in which the displayable objects are rendered. Studio attributes include renderering control classes, camera projection classes and light attribute classes.

Rendering Control Classes

Some of the rendering control classes, listed in Table D1-4, are inappropriate for some styles of renderers. For example, a renderer that does not use raytracing would not use the *DoGlbRendRayLevel* attribute. The renderer specific documentation describes which rendering control classes are supported by each renderer.

Table D1-4. Rendering Control Classes

DoGlbRendMaxObjs	DoSampleJitter
DoGlbRendMaxSub	DoSampleJitterSwitch
DoGlbRendRayLevel	DoSampleSuper
DoSampleAdaptive	DoSampleSuperSwitch
DoSampleAdaptiveSwitch	DoStereo
DoSampleFilter	DoStereoSwitch

Camera Projection Classes

The camera projection classes specify the type of projection matrix to be used when generating the 2–D image of the 3–D space. Most renderers can use all the camera projection classes listed in Table D1-5. See the renderer specific documentation for the list of camera projection classes actually supported.

Table D1-5. Camera Projection Classes

DoCameraMatrix	DoPerspective
DoParallel	DoProjection

Light Attribute Classes

The light attribute classes define the characteristics of the lights used in a scene. Table D1-6 lists the full set of light attribute classes. Not all renderers simulate all light characteristics. Most renderers use light color, light intensity and several of the light types. See the renderer specific documentation for the attributes supported by a particular renderer.

Table D1-6. Light Attribute Classes

DoLightAttenuation	DoLightType
DoLightColor	*DcLightAmbient*
DoLightIntens	*DcLightInfinite*
DoLightSpreadAngles	*DcLightPoint*
DoLightSpreadExp	*DcLightPointAttn*
	DcLightSpot
	DcLightSpotAttn

Some renderers do not use the light attribute parameters exactly as defined. The attributes that are most likely to exhibit usage variations are: *DoLightAttenuation, DoLightSpreadAngles,* and *DoLightSpreadExp.* The renderer specific documentation describes renderer specific variations.

Geometric Transform Classes

The geometric transform classes modify the current transformation matrix. Table D1-7 lists the geometric transformation classes. These classes are a standard part of Doré and are supported by all renderers.

Table D1-7. Geometric Transform Classes

DoLookAtFrom	DoRotate	DoTransformMatrix
DoPopMatrix	DoScale	DoTranslate
DoPushMatrix	DoShear	

Primitive Classes

The Doré primitive objects are displayable objects. Doré has a wide variety of graphics primitives. Most renderers can directly render only a subset of the Doré primitives. These are referred to as the *base* primitives for a given renderer. Most Doré primitives also possess a standard alternate representation. If the alternate representation for a primitive can be drawn by a renderer, then that renderer can, if necessary, render the primitive using the alternate representation.

For example, a renderer that cannot directly draw spheres will render the standard alternate representation for a sphere, which is composed of a mesh of triangles.

Table D1-8 lists the current set of primitive object classes and their standard alternate representations.

Table D1-8. Primitive Classes

Primitive Class	Alternate Representations
DoLineList	
DoAnnoText	DoLineList/DoTriangleList
DoNURBSurf	DoTriangleList
DoPatch	DoTriangleList
DoPointList	
DoPolygon	DoTriangleList
DoPolygonMesh	DoTriangleList
DoPolyline	
DoPolymarker	DoLineList/DoTriangleList
DoPrimSurf	DoTriangleList/DoTriangleMesh
DoSimplePolygon	DoLineList/DoTriangleList
DoSimplePolygonMesh	DoLineList/DoTriangleMesh
DoText	DoLineList/DoTriangleList
DoTorus	DoTriangleList
DoTriangleList	
DoTriangleMesh	
DoVarLineList	
DoVarPointList	
DoVarSimplePolygonMesh	DoVarLineList/DoVarTriangleList
DoVarTriangleMesh	

Primitive Attribute Classes

Primitive attribute classes constitute a large part of the Doré system. Many primitive attribute classes relate directly to rendering. The renderer specific documentation lists the attributes that affect the operation of each renderer.

Shading Classes

The shading attribute classes listed in Table D1-9 control the shading parameters for subsequently executed primitive objects. Some shading attributes, such as diffuse color, are supported by most renderers. Many other shading attributes, such as refraction, depend on advanced shading features not supported by all renderers.

Texture Map Classes

The texture map classes are shading classes that modify the standard shading attributes as a function of a parameterization of a primitive's surface. Table D1-10 lists the texture map classes.

Table D1-9. Shading Classes

DoAmbientIntens	DoShadowSwitch
DoAmbientSwitch	DoSpecularColor
DoDepthCue	DoSpecularFactor
DoDepthCueSwitch	DoSpecularIntens
DoDiffuseColor	DoSpecularSwitch
DoDiffuseIntens	DoSurfaceShade
DoDiffuseSwitch	DoTranspColor
DoInterpType	DoTranspIntens
DoLightSwitch	DoTranspSwitch
DoReflectionSwitch	DoTranspOrientColor
DoRefractionIndex	DoTranspOrientExp
DoRefractionSwitch	DoTranspOrientIntens
DoShadeIndex	DoTranspOrientSwitch

Table D1-10. Texture Map Classes

DoTextureMapBump	DoTextureMapBumpSwitch
DoTextureMapDiffuseColor	DoTextureMapDiffuseColorSwitch
DoTextureMapEnviron	DoTextureMapEnvironSwitch
DoTextureMapTranspIntens	DoTextureMapTranspIntensSwitch

Texture Attribute Classes

The current texture attribute values are bound to a texture map object during traversal. The texture map object is itself an attribute applying to primitive objects. Table D1-11 lists the texture attribute classes. Certain renderers many not be able to use all texture attributes, even if they do support some texture mapping. For example, a renderer that supports only 2–D texture mapping will ignore all 3–D texture attributes.

Table D1-11. Texture Attribute Classes

DoTextureAntiAlias	DoTextureScaleUVW
DoTextureExtendUV	DoTextureTranslateUV
DoTextureExtendUVW	DoTextureTranslateUVW
DoTextureOp	DoTextureUVIndex
DoTextureScaleUV	DoTextureUVWIndex

Display Control Classes

The display control classes control certain renderer characteristics. These are listed in Table D1-12.

Table D1-12. Display Control Classes

DoBackfaceCullSwitch	DoClipVol	DoLineType
DoBackfaceCullable	DoHiddenSurfSwitch	DoLineWidth
DoClipSwitch	DoInvisSwitch	DoRepType

Traversal Control Classes

The traversal control classes shown in Table D1-13 are a standard part of Doré. These classes affect the Doré database traversal process but are not directly used by the renderer. These classes are renderer independent.

Table D1-13. Traversal Control Classes

DoBoundingVol	DoFilter
DoBoundingVolSwitch	DoMinBoundingVolExt
DoExecSet	DoNameSet

Picking Attribute Classes

The picking attributes, shown in Table D1-14, are renderer independent.

Table D1-14. Picking Attribute Classes

DoPickID	DoPickSwitch

Alternate Representation Attribute Classes

The attribute classes shown in Table D1-15 are used by the primitives when generating their alternate representations. These attributes are renderer independent.

Table D1-15. Alternate Representation Attribute Classes

DoGenerateTextureUV	DoSubDivSpec

Marker Attribute Classes

The marker attributes, listed in Table D1-16, apply to the *DoPolymarker* primitive. Most renderers use the standard alternate object to render markers. In that case, the *DoPolymarker* primitive uses the marker attributes to convert the marker description to either lines or triangles. When the standard alternate representation is used the marker attributes are renderer independent.

Table D1-16. Marker Attribute Classes

DoMarkerFont	DoMarkerGlyph	DoMarkerScale

Text Attribute Classes

The text attributes, listed in Table D1-17, pertain to the *DoText* and *DoAnnoText* primitive classes. Most renderers use the standard alternate object to render these primitives. In that case, the primitive uses the text attributes to decompose the text description into either lines or triangles. When the standard alternate representation is used the text attributes are renderer independent.

Table D1-17. Text Attribute Classes

DoTextAlign	DoTextHeight	DoTextSpace
DoTextExpFactor	DoTextPath	DoTextUpVector
DoTextFont	DoTextPrecision	

Primitive Update Functions

The primitive update functions, listed in Table D1-18, are renderer independent. They are used to notify the primitive of user data changes.

Table D1-18. Primitive Update Functions

DpUpdVarLineList	DpUpdVarSimplePolygonMesh
DpUpdVarPointList	DpUpdVarTriangleMesh

Extension Functions

The extension functions, listed in Table D1-19, are for implementing user–defined primitives. They are renderer independent.

Table D1-19. Extension Functions

DeAddClass	DeExecuteAlternate	DeInqPickable
DeCreateObject	DeInitializeObjPick	DeInqRenderable
DeDeleteObject		

Device Functions

Doré has many functions that operate on device objects. These functions, listed in Table D1-20, are renderer independent.

Table D1-20. Device Functions

DdInqColorEntries	DdInqViewport
DdInqColorTableSize	DdInqVisualType
DdInqExtent	DdPick
DdInqFonts	DdPickObjs
DdInqFrame	DdSetColorEntries
DdInqNumFonts	DdSetFrame
DdInqPickAperture	DdSetPickAperture
DdInqPickCallback	DdSetPickCallback
DdInqPickPathOrder	DdSetPickPathOrder
DdInqPixelData	DdSetShadeMode
DdInqResolution	DdSetShadeRanges
DdInqShadeMode	DdSetViewport
DdInqShadeRanges	DdUpdate

Frame Functions

Doré has many functions that operate on frame objects. These functions, listed in Table D1-21, are renderer independent.

Table D1-21. Frame Functions

DfInqBoundary	DfInqViewGroup	DfSetJust
DfInqJust	DfSetBoundary	DfUpdate

View Functions

Doré has many functions that operate on view objects. These functions, listed in Table D1-22, are renderer independent.

Table D1-22. View Functions

DvInqActiveCamera	DvSetActiveCamera
DvInqBackgroundColor	DvSetBackgroundColor
DvInqBoundary	DvSetBoundary
DvInqClearFlag	DvSetClearFlag
DvInqDefinitionGroup	DvSetRendStyle
DvInqDisplayGroup	DvSetShadeIndex
DvInqRendStyle	DvSetUpdateType
DvInqShadeIndex	DvUpdate
DvInqUpdateType	

Group Functions

Doré has many functions that operate on group objects. These functions, listed in Table D1-23, are renderer independent.

Table D1-23. Group Functions

DgAddObj	DgInqObjAtPos
DgAddObjToGroup	DgInqOpen
DgCheck	DgInqSize
DgClose	DgOpen
DgDelEle	DgReplaceObj
DgDelEleBetweenLabels	DgReplaceObjInGroup
DgDelEleRange	DgSetElePtr
DgEmpty	DgSetElePtrRelLabel
DgInqElePtr	

System Functions

Doré has many system functions. These functions are renderer independent.

DYNAMIC RENDERER

APPENDIX D2

Overview

The goal of the Dynamic Renderer is to display images as quickly as possible so the user can interact with the models in real time. To this end, it is designed to allow the associated device driver to take total advantage of available graphics hardware. For the actual rendering, the Dynamic Renderer relies on the graphics pipeline of the device being accessed. The pipeline usually includes transformation, clipping, scan conversion and shading operations. Different devices have different pipeline implementations, so the actual output from the Dynamic Renderer varies from device to device.

Device Driver Support

The Dynamic Renderer requires the Dynamic Renderer Output Module Interface in addition to the Device Control Module Interface. See the appendix on device drivers for more information.

Attribute Classes

The Dynamic Renderer passes most of the Doré attributes to the device driver. See *Appendix F: Doré Device Drivers* for specific information on each device driver, including which attributes are supported.

Table D2-1 lists the attributes that are currently *not* passed to the device by the Dynamic Renderer.

Table D2-1. Classes Not Available with the Dynamic Renderer

DoGlbRendMaxObjs	DoSampleSuper
DoGlbRendMaxSub	DoSampleSuperSwitch
DoGlbRendRayLevel	DoRefractionIndex
DoSampleAdaptive	DoRefractionSwitch
DoSampleAdaptiveSwitch	DoTranspOrientColor
DoSampleFilter	DoTranspOrientExp
DoSampleJitter	DoTranspOrientIntens
DoSampleJitterSwitch	DoTranspOrientSwitch

STANDARD PRODUCTION RENDERER

APPENDIX D3

Overview

The Doré Standard Production Renderer is a spatial subdivision raytracer which performs all rendering calculations using portable machine independent software. It has the ability to generate images with transparency, reflections, and shadows, along with many of the other standard Doré attributes. It does not support texture mapping, image antialiasing, or vertex colors for primitives.

Not all renderers are able to use all the Doré attributes or to render all Doré primitives. This section lists the Doré classes that may be handled differently by different renderers, and indicates which ones are used by the Standard Production Renderer. If a scene includes objects that the Standard Production Renderer cannot use (such as a *DoRefractionIndex* object), those objects are ignored and do not affect the image generation.

This appendix does not list all Doré object classes. The renderer overview at the front of this appendix lists all object classes and indicates which classes are available for all renderers.

Device Driver Support

The Standard Production Renderer requires only the Production Renderer Output Module Interface, in addition to the Device Control Module Interface. See *Appendix F: Doré Device Drivers* for more information on device drivers.

Studio Attribute Classes

The studio attribute classes are classes of objects that affect the global environment in which displayable objects are rendered. These include rendering control classes, camera attributes that control projection matrices, and lighting attributes.

Rendering Control Classes

Doré includes attributes to control the rendering process. Table D3-1 lists the rendering control attributes that are used by the Standard Production Renderer. Antialiasing and stereo are not supported.

Table D3-1. Rendering Control Attributes

Attribute Class	Available
DoGlbRendMaxObjs	yes
DoGlbRendMaxSub	yes
DoGlbRendRayLevel	yes
DoSampleAdaptive	
DoSampleAdaptiveSwitch	
DoSampleFilter	
DoSampleJitter	
DoSampleJitterSwitch	
DoSampleSuper	
DoSampleSuperSwitch	
DoStereo	
DoStereoSwitch	

Camera Projection Classes

Table D3-2 lists the types of camera projection matrices that the Standard Production Renderer supports.

Table D3-2. Camera Projection Attributes

Attribute Class	Available
DoCameraMatrix	
DoParallel	yes
DoPerspective	yes
DoProjection	

Light Attribute Classes

Table D3-3 lists all the lighting attributes in the Doré system, and indicates which attributes are used by the Standard Production Renderer.

Table D3-3. Lighting Attributes

Attribute Class	Available
DoLightAttenuation	
DoLightColor	yes
DoLightIntens	yes
DoLightSpreadAngles	
DoLightSpreadExp	
DoLightType	
DcLightAmbient	yes
DcLightInfinite	yes
DcLightPoint	
DcLightPointAttn	
DcLightSpot	
DcLightSpotAttn	

Primitive Classes

The following is a complete list of Doré primitive object classes. Table D3-4 lists which of the Doré primitives the Standard Production Renderer can render directly and which ones are rendered using alternate representations.

Table D3-4. Doré Primitives

Primitive Class	Direct	Alternate Obj
DoAnnoText		
DoLineList		
DoNURBSurf		yes
DoPatch		yes
DoPointList		
DoPolygon		yes
DoPolygonMesh		yes
DoPolyline		
DoPolymarker		
DoPrimSurf		
DcSphere	yes	
DcCylinder	yes	
DcBox	yes	
DcCone	yes	
DoSimplePolygon		yes
DoSimplePolygonMesh		yes

Table D3-4. Doré Primitives (continued)

Primitive Class	Direct	Alternate Obj
DoText		yes(1)
DoTorus	yes	
DoTriangleList	yes	
DoTriangleMesh	yes	
DoVarLineList		
DoVarPointList		
DoVarSimplePolygonMesh		yes
DoVarTriangleMesh	yes	

(1) Only polygonal fonts like DcHelvetica.

Primitive Attribute Classes

The primitive attributes are used by the renderer to affect the appearance of a primitive when it is displayed.

Shading Attribute Classes

Table D3-5 lists the shading attributes, and indicates which attributes are used by the Standard Production Renderer.

Table D3-5. Shading Attributes

Attribute Class	Available
DoAmbientIntens	yes
DoAmbientSwitch	yes
DoDepthCue	
DoDepthCueSwitch	
DoDiffuseColor	yes
DoDiffuseIntens	yes
DoDiffuseSwitch	yes
DoInterpType	
DcConstantShade	
DcVertexShade	
DcSurfaceShade	yes
DoLightSwitch	
DoReflectionSwitch	yes
DoRefractionIndex	
DoRefractionSwitch	

Table D3-5. Shading Attributes (continued)

Attribute Class	Available
DoShadeIndex	
DoShadowSwitch	yes
DoSpecularColor	yes
DoSpecularFactor	
DoSpecularIntens	yes
DoSpecularSwitch	yes
DoSurfaceShade	
DcShaderConstant	
DcShaderLightSource	yes
DoCallback	
DoTranspColor	yes
DoTranspIntens	yes
DoTranspSwitch	yes
DoTranspOrientSwitch	
DoTranspOrientColor	
DoTranspOrientIntens	
DoTranspOrientExp	

Texture Map Classes

The Standard Production Renderer does not support texture mapping, thus none of the texture map classes are used.

Texture Attribute Classes

The Standard Production Renderer does not support texture mapping, thus none of the texture attributes are used.

Marker Attribute Classes

The Standard Production Renderer does not display objects that are in frame coordinates. The marker attributes affect *DoPolymarker* primitives which is in frame coordinates, so these attributes are effectively ignored.

Text Attribute Classes

The text attributes are used by the *DoText* and *DoAnnoText* primitives. Like *DoPolymarker, DoAnnoText* is a primitive that uses frame coordinates, so it is not supported. For text, the Standard Production Renderer uses the standard alternate representation,

so all text attributes are effective. Note that since the Renderer does not support lines, only polygonal fonts are displayed.

Display Control Classes

Table D3-6 shows the display control attributes that are used by the Standard Production Renderer.

Table D3-6. Display Control Attributes

Attribute Class	Available
DoBackfaceCullSwitch	yes
DoBackfaceCullable	yes
DoClipSwitch	
DoClipVol	
DoHiddenSurfSwitch	
DoInvisSwitch	yes
DoLineType	
DoLineWidth	
DoRepType	
DcPoints	
DcWireframe	
DcSurface	yes
DcOutlines	

MEDIA LOGIC RENDERER

APPENDIX D4

Overview

The Media Logic Renderer is a high-quality renderer that uses a combination of scanline and raytracing techniques for image production. The Media Logic Renderer supports a broad set of shading attributes. In addition to utilizing the more common Doré attributes, it has the ability to generate images with transparency, reflections, refractions, texture mapping and image antialiasing.

Not all renderers are able to use all the Doré attributes or render all the Doré primitives. This section lists the Doré object classes that may be handled differently by different renderers and indicates which ones are used by the Media Logic renderer when generating an image. If a scene includes primitives and attributes that the Media Logic renderer does not support, those objects are ignored and do not affect image generation.

This section does not list all Doré object classes. The appendix on Doré Renderers lists all object classes and indicates which classes are available for all renderers.

Device Driver Support

The Media Logic Renderer requires only the Production Renderer Output Module Interface and the Device Control Module Interface. See *Appendix F: Doré Device Drivers* for more details on the specifics of device drivers.

Studio Attribute Classes

The studio attribute classes affect the global environment in which displayable objects are renderered. These include rendering control for sampling and antialiasing, camera attributes that control projection matrices, and lighting attributes.

Rendering Control Classes

Doré includes attributes to control the rendering process. Table D4-1 lists the rendering control attributes that are used by the Media Logic Renderer.

Table D4-1. Rendering Control Attributes

Attribute Class	Available
DoGlbRendMaxObjs	
DoGlbRendMaxSub	
DoGlbRendRayLevel	yes
DoSampleAdaptive	yes
DoSampleAdaptiveSwitch	yes
DoSampleFilter	
DoSampleJitter	yes
DoSampleJitterSwitch	yes
DoSampleSuper	yes
DoSampleSuperSwitch	yes
DoStereo	
DoStereoSwitch	

Camera Projection Classes

Table D4-2 lists the types of camera projection matrices that the Media Logic Renderer supports.

Table D4-2. Camera Projection Attributes

Attribute Class	Available
DoCameraMatrix	yes
DoParallel	yes
DoPerspective	yes
DoProjection	yes

Light Attribute Classes

Table D4-3 lists all the lighting attributes in the Doré system. They are all used by the Media Logic Renderer.

Table D4-3. Lighting Attributes

Attribute Class	Available
DoLightAttenuation	yes
DoLightColor	yes
DoLightIntens	yes
DoLightSpreadAngles	yes
DoLightSpreadExp	yes
DoLightType	
DcLightAmbient	yes
DcLightInfinite	yes
DcLightPoint	yes
DcLightPointAttn	yes
DcLightSpot	yes
DcLightSpotAttn	yes

Primitive Classes

The following is a complete list of Doré primitive object classes. Table D4-4 shows which Doré primitives the Media Logic Renderer can render directly and which are rendered using alternate representations.

Table D4-4. Doré Primitives

Primitive Class	Direct	Alternate Obj
DoAnnoText		
DoLineList	yes(1)	
DoNURBSurf		yes
DoPatch		yes
DoPointList	yes(1)	
DoPolygon		yes
DoPolygonMesh		yes
DoPolyline	yes(1)	
DoPolymarker		
DoPrimSurf		
DcSphere		yes
DcCylinder		yes
DcBox		yes
DcCone		yes
DoSimplePolygon		yes
DoSimplePolygonMesh		yes
DoText		yes
DoTorus		yes
DoTriangleList	yes	
DoTriangleMesh	yes	
DoVarLineList	yes(1)	
DoVarPointList	yes(1)	
DoVarSimplePolygonMesh		yes
DoVarTriangleMesh	yes	

(1) Lines and points are not raytraced.

Primitive Attribute Classes

The primitive attributes affect the appearance of primitives when they are displayed.

Shading Attribute Classes

The Media Logic Renderer supports a broad set of shading attributes. Table D4-5 lists the Doré shading attributes, and indicates which attributes are supported by the Media Logic Renderer.

Texture Map Classes

Table D4-6 lists the texture map classes that are used by the Media Logic Renderer. All the texture map attributes are supported.

Table D4-5. Shading Attributes

Attribute Class	Available
DoAmbientIntens	yes
DoAmbientSwitch	yes
DoDepthCue	
DoDepthCueSwitch	
DoDiffuseColor	yes
DoDiffuseIntens	yes
DoDiffuseSwitch	yes
DoInterpType	
DcConstantShade	
DcVertexShade	
DcSurfaceShade	yes
DoLightSwitch	yes
DoReflectionSwitch	yes
DoRefractionIndex	yes
DoRefractionSwitch	yes
DoShadeIndex	
DoShadowSwitch	yes
DoSpecularColor	yes
DoSpecularFactor	yes
DoSpecularIntens	yes
DoSpecularSwitch	yes
DoSurfaceShade	
DcShaderConstant	
DcShaderLightSource	yes
DoCallback	
DoTranspColor	yes
DoTranspIntens	yes
DoTranspSwitch	yes
DoTranspOrientSwitch	yes
DoTranspOrientColor	yes
DoTranspOrientIntens	yes
DoTranspOrientExp	yes

The texture map attributes (except for the switches) take 3 parameters.

operator
: the Media Logic Renderer supports both *DcMapReplace* and *DcMapAdd*. This means you can have multiple, overlapping textures of the same type.

mapping
: the Media Logic Renderer supports 2–D table-lookup texture

Table D4-6. Texture Map Classes

Attribute Class	Available
DoTextureMapBumpSwitch	yes
DoTextureMapBump	yes
DoTextureMapDiffuseColorSwitch	yes
DoTextureMapDiffuseColor	yes
DoTextureMapEnvironSwitch	yes
DoTextureMapEnviron	yes
DoTextureMapTranspIntensSwitch	yes
DoTextureMapTranspIntens	yes

mapping only. For mapping diffuse color and transparent intensity use *DcStdTableLookup*. For bump mapping use *DcStdBumpMap*. For environment mapping use *DcStdSphereEnvironMap*.

raster

for the map data, the Media Logic Renderer can use either standard Doré raster format (file or memory) or one of several special file types (see *DoFileRaster man* page). If the raster is in standard Doré format, the data channels are used as described in the *man* pages for the texture map attributes. Four special file types are supported. They are *"MLR art"* (Media Logic Artisan file), *"MLR tmap"* (Media Logic MIP-map texture file), *"MLR rmap"* (Media Logic MIP-map roughness file), and *"MLR rle"* (Media Logic runlength-encoded image file). Contact Media Logic Incorporated for more information on these file formats.

Unlike most other primitive attributes the texture map classes also inherit attributes themselves. The texture attribute classes are discussed in the following subsection.

Texture Attribute Classes

The current texture attribute values are bound to texture map objects during traversal. Table D4-7 lists the texture attributes that apply to the Media Logic Renderer. The Media Logic Renderer only supports 2–D texture mapping, and ignores 3–D texture attributes.

Table D4-7. Texture Attributes

Attribute Class	Available
DoTextureAntiAlias	yes
DoTextureExtendUV	yes
DoTextureExtendUVW	
DoTextureOp	
DcTextureReplace	yes
DcTextureMultiply	yes
DcTextureBlend	yes
DcTextureAdd	yes
DoTextureScaleUV	yes
DoTextureScaleUVW	
DoTextureTranslateUV	yes
DoTextureTranslateUVW	
DoTextureUVIndex	yes
DoTextureUVWIndex	

Display Control Classes

Table D4-8 shows the display control attributes that are used by the Media Logic Renderer.

Table D4-8. Display Control Attributes

Attribute Class	Available
DoBackfaceCullSwitch	yes
DoBackfaceCullable	yes
DoClipSwitch	
DoClipVol	
DoHiddenSurfSwitch	
DoInvisSwitch	yes
DoLineType	yes
DoLineWidth	yes
DoRepType	
DcPoints	
DcWireframe	
DcSurface	yes
DcOutlines	yes

Marker Attribute Classes

The Media Logic renderer does not display objects that are in frame coordinates. The marker attributes affect the *DoPolymarker* primitive, which is in frame coordinates, so these attributes are

effectively ignored.

Text Attribute Objects

The text attributes are used by the *DoText* and *DoAnnoText* primitives. Like *DoPolymarker, DoAnnoText* is a primitive that uses frame coordinates, so it is not displayed. For text primitives, the Media Logic Renderer uses the standard alternate representation. All text attributes are effective.

DORÉ CONFIGURATIONS

APPENDIX E

Contents of this Appendix

This appendix contains the following subsections:

Overview

Doré configurations vary, depending on the hardware on which the configuration runs, and on the port of Doré to that hardware platform. Different Doré configurations utilize different device drivers and renderers.

This appendix details the different Doré configurations available from Stardent Computer. Some of these configurations are only available with the Portable Doré product.

Each section briefly describes the hardware on which a particular configuration runs. They describe the device drivers included in each configuration, and describe how to interface correctly with available renderers. Some of the renderers come standard with the Doré, while other renderers are optional. Each section also describes any other details that are pertinent to each configuration.

STANDARD X11

APPENDIX E1

Overview

This section describes the specifics of running Doré on an X11 platform, using the standard X11 protocol. Note that this configuration, unlike most others, is not tied to a particular hardware platform. It will run on most devices that support the X11 protocol. It therefore does not take advantage of any graphics hardware support that might be available on the device, and will be less efficient than a configuration that was tailored specifically to the device. For example, the standard X11 configuration will run on the Stardent 3000, but it will be much slower than the Stardent 3000 configuration, even though that configuration also uses X11 windows.

This section includes a list of device drivers supported by the standard X11 configuration of Doré, and a list of available renderers.

The Standard X11 Doré configuration can be built from Portable Doré source Release 2.2 or above. This configuration corresponds to the **stdx** Portable Doré system configuration.

Device Drivers

Two device drivers are provided with this configuration of Doré. They are: the Rasterfile device driver, and the Standard X11 device driver. Each of these device drivers is briefly described below. For more information on the device drivers see *Appendix F: Doré Device Drivers*.

Rasterfile Device Driver

The Rasterfile device driver is accessed from Doré applications by creating a device of type *"rasterfile"* in a call to *DoDevice*. For example:

```
device = DoDevice ("rasterfile", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Rasterfile device driver are:

–width *size*
–height *size*
–filename *name*

Standard X11 Device Driver

The Standard X11 device driver is accessed from Doré applications by creating a device of type *"stdx11"* in a call to *DoDevice*. For example:

```
device = DoDevice ("stdx11", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Standard X11 device driver are:

–singlebuffered
–geometry *geomstring*
–visualtype [*DcPseudoColor* | *DcStaticGrey*]
–window *xwindow*
–display *xdisplay*
–zbuffer

Renderers

The X11 version of Doré includes two renderers. They are the Dynamic Renderer and the Standard Production Renderer. The Dynamic Renderer is installed so that it is used when the render style (*DvSetRendStyle*) is *DcRealTime*. The Standard Production Renderer is installed so that it is used when the render style is *DcProductionTime*.

For more details on which Doré attributes and primitives are used by these renderers see *Appendix D: Doré Renderers* in this manual. The Dynamic Renderer depends heavily on the particular device driver being used. For more information about the capabilities of device drivers see *Appendix F: Doré Device Drivers*.

STARDENT 1500/3000

APPENDIX E2

Overview

This section describes the specifics of running Doré on the Stardent 1500/3000 graphics supercomputers (also known as the TITAN). Both machines are double-buffered, 24-bit color workstations manufactured by Stardent Computer, and both run the X11+ windowing system.

This section includes a list of the device drivers supported by the Stardent 1500/3000 configuration of Doré, a list of available renderers and other information particular to Doré on the Stardent 1500/3000 supercomputers.

The Stardent 1500/3000 Doré configuration is a part of the standard software release on Stardent 1500/3000 machines.

Device Drivers

Three device drivers are provided with the Stardent 1500/3000 configuration of Doré. They are: the Rasterfile device driver, the Standard X11 device driver, and the Stardent 1500/3000 device driver. Each of these device drivers is briefly described below. For more information on these device drivers see *Appendix F: Doré Device Drivers*.

Rasterfile Device Driver

The Rasterfile device driver is accessed from Doré applications by creating a device of type *"rasterfile"* in a call to *DoDevice*. For example:

```
device = DoDevice ("rasterfile", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Rasterfile device driver are:

–width *size*
–height *size*
–filename *name*

Standard X11 Device Driver

The Standard X11 device driver is accessed from Doré applications by creating a device of type *"stdx11"* in a call to *DoDevice*. For example:

```
device = DoDevice ("stdx11", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Standard X11 device driver are:

–singlebuffered
–geometry *geomstring*
–visualtype [*DcPseudoColor* | *DcStaticGrey*]
–window *xwindow*
–display *xdisplay*
–zbuffer

Stardent 1500/3000 Device Driver

The Stardent 1500/3000 device driver is accessed from Doré applications by creating a device of type *"ardentx11"* or *"stardentx11"* in a call to *DoDevice*. For example:

```
device = DoDevice ("stardentx11", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Stardent 1500/3000 device driver are:

–singlebuffered
–geometry *geomstring*
–visualtype [*DcDirectColor* | *DcPseudoColor*]
–window *xwindow*
–display*xdisplay*
–stereo

Renderers

The Stardent 1500/3000 configuration of Doré includes two renderers. They are the Dynamic Renderer and the Standard Production Renderer. The Dynamic Renderer is installed so that it is

used when the render style (*DvSetRendStyle*) is *DcRealTime*. The Standard Production Renderer is installed so that it is used when the render style is *DcProductionTime*.

In addition to the two renderers that come with the Stardent 1500/3000 configuration, a third renderer, the Media Logic Renderer, is optionally available for Stardent 1500/3000 systems. This renderer is a high quality renderer that provides additional capabilities over those of the Standard Production Renderer. Its features include: texture mapping, image antialiasing, and in-scene light sources.

For more details on which Doré attributes and primitives are used by these renderers see *Appendix D: Doré Renderers* in this manual. The image quality and speed of the Dynamic Renderer depends heavily on the particular device driver being used. For more information about the capabilities of the device drivers see *Appendix F: Doré Device Drivers*.

Doré Libraries

Doré provides several versions of the Doré library for use with Stardent Computer Inc. computers. The libraries available vary from machine to machine. The library names also vary, so exercise caution when choosing a library. Separate shared and unshared libraries are provided for single-precision Doré and double-precision Doré. See the following subsection, *Selecting a Library*, for more details on how to activate the different libraries.

Shared libraries allow more than one independent process to use the same object libraries rather than replicating them for each process.

Single-precision libraries use single precision floating-point numbers to exchange information with Doré application programs. Double-precision libraries use double-precision floating-point numbers to exchange information with Doré application programs.

Stardent 1500 Libraries

The Stardent 1500 configuration provides shared and unshared double precision versions of the Doré library. A single precision library is not available for the Stardent 1500.

The default library for the Stardent 1500 is a double-precision unshared library. The two libraries available on the Stardent 1500 are:

libUdore_d.a
: unshared library with the double-precision Doré application program interface (default library for Stardent 1500)

libSdore_d.a
: shared library with the double-precision Doré application program interface

For compatibility, *libdore_d.a* is supplied as a symbolic link to *libUdore_d.a*. Thus, you can link the default library with *–ldore_d*.

Stardent 3000 Libraries

The Stardent 3000 Configuration provides both shared and unshared versions of the Doré library. On the Stardent 3000 separate libraries are also provided for single-precision Doré and double-precision Doré. Unlike the default Stardent 1500 library, the Stardent 3000 default is a shared single-precision library. The four libraries provided on Stardent 3000 systems are:

libSdore.a
: shared library with the single-precision Doré application program interface (default library for Stardent 3000)

libSdore_d.a
: shared library with the double-precision Doré application program interface

libUdore.a
: unshared library with the single-precision Doré application program interface

libUdore_d.a
: unshared library with the double-precision Doré application program interface

On the Stardent 3000, *libdore.a* is included as a symbolic link to *libSdore.a* and *libdore_d.a* as a symbolic link to *libSdore_d.a*. Thus, you can link the default library with *–ldore*.

Selecting a Library

When compiling Doré programs on the Stardent 3000 the *dore.h* include file must be configured to select either the double-precision or the single-precision floating point interface. This is accomplished with a compile-time flag. In addition, the correct library should be linked to the application program.

The compiler option *-DDORE_REAL_SINGLE* should be used to configure *dore.h* for the single-precision interface (Stardent 3000 only). The compiler option *-DDORE_REAL_DOUBLE* configures *dore.h* for the double-precision interface. If no compiler option is specified, double-precision is assumed. The application must then be linked with a double-precision library.

To build with the single-precision shared library on the Stardent 3000, compile and load with a command similar to:

```
cc -DDORE_REAL_SINGLE my_prog.c -o my_prog -ldore -lXd \
        -lXtitan -lXB -lX11 -lm
```

To build with the double-precision shared library on the Stardent 3000, compile and load with a command similar to:

```
cc -DDORE_REAL_DOUBLE my_prog.c -o my_prog -ldore_d -lXd \
        -lXtitan -lXB -lX11 -lm
```

or just:

```
cc my_prog.c -o my_prog -ldore_d -lXd \
        -lXtitan -lXB -lX11 -lm
```

ANSI C Prototyping

The Stardent 1500/3000 configuration of Doré provides ANSI C function prototyping. It is important not to declare Doré functions in your code. All Doré functions are declared by *dore.h*. *dore.h* properly configures Doré for double- or single-precision floating-point operation, and redeclaring Doré functions could corrupt these declarations.

Multiprocessor Doré

The Stardent 1500/3000 device driver is able to use multiple processors when the Dynamic Renderer is used. For primitives containing large numbers of elements, groups of elements are rendered in parallel, providing a significant performance

enhancement.

Enable multiprocessor Doré by specifying the number of processors when Doré is initialized with *DsInitializeSystem*. For example:

```
DsInitializeSystem (4);
```

Doré on the Stardent 1500/3000 allows values from zero to four.

A value of zero instructs Doré to use only one processor. A value greater than zero will result in multiprocessing with that many processors, or the number of processors installed in the system, whichever is less. Note that a value of one is only useful for debugging, since it incurs the overhead of multiprocessing while using only one processor. For this reason, use a value of zero to specify single-processor Doré.

GENERIC SUN

APPENDIX E3

Overview

This section describes the specifics of Doré running on a generic Sun, that is, an 8–bit, 12–bit, or 24–bit color workstation without graphics hardware accelerators, manufactured by Sun Microsystems.

This section includes a list of device drivers supported by the generic Sun configuration of Doré, a list of available renderers, and other information particular to Doré running on a generic Sun.

The Generic Sun Doré configuration can be built from Portable Doré source Release 2.1 or above. This configuration corresponds to the **sungen** Portable Doré system configuration.

Device Drivers

Two device drivers are provided with the generic Sun configuration of Doré: the Rasterfile device driver, and the Generic SunView device driver. Each of these device drivers is briefly described below. For more information on the device drivers see *Appendix F: Doré Device Drivers.*

Rasterfile Device Driver

The Rasterfile device driver is accessed from Doré applications by creating a device of type *"rasterfile"* in a call to *DoDevice.* For example:

```
device = DoDevice ("rasterfile", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Rasterfile device driver are:

–width *size*

–height *size*
–filename *name*

Generic SunView Device Driver

The Generic SunView device driver is accessed from Doré applications by creating a device of type *"sunview"* in a call to *DoDevice*. For example:

```
device = DoDevice ("sunview", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Generic SunView device driver are:

–width *size*
–height *size*
–canvas *canvasid*

Renderers

The generic Sun configuration of Doré includes two renderers. They are the Dynamic Renderer and the Stardent Production Renderer. The Dynamic Renderer is installed so that it is used when the render style (*DvSetRendStyle*) is set to *DcRealTime*. The Stardent Production Renderer is installed so that it is used when the render style is set to *DcProductionTime*.

For more details on which Doré attributes and primitives are used by these renderers see the appendix on Doré Renderers. The Dynamic Renderer depends heavily on the particular device driver being used, for more information about the capabilities of the device drivers see the appendix on Doré Device Drivers. At this time neither of the standard devices supports the Dynamic Renderer, so dynamic rendering is not possible with this configuration.

Floating Point Precision

Doré allows both a single precision and a double precision application interface for Doré real numbers (*DtReal*). A library can be built from Portable Doré source to support either precision for Doré real numbers. By convention the single precision library is *libdore.a* and the double precision library is *libdore_d.a*. The include file *dore.h* allows compile time selection of the floating point precision.

To use single precision, files should be compiled with the *-DDORE_REAL_SINGLE*, and linked with *libdore.a*, for example:

```
cc -DDORE_REAL_SINGLE test.c -o test -ldore -lsuntool \
        -lsunwindow -lpixrect -lm
```

To use double precision, files should be compiled with the *-DDORE_REAL_DOUBLE*, and linked with *libdore_d.a* for example:

```
cc -DDORE_REAL_DOUBLE test.c -o test -ldore_d -lsuntool \
        -lsunwindow -lpixrect -lm
```

SUN/CXP

APPENDIX E4

Overview

This section describes the specifics of running Doré on the Sun Microsystems Sun-3/CXP or Sun-4/CXP workstation. These machines are 8–bit double–buffered systems that have hardware support for 3–D matrix transformations and Gouraud shading.

This section includes a list of device drivers supported by the Sun/CXP configuration of Doré, a list of available renderers, and other information particular to running Doré on the Sun/CXP.

The Sun/CXP Doré configuration can be built from Portable Doré source Release 2.1 or above. This configuration corresponds to the **suncxp** Portable Doré system configuration.

Device Drivers

Three device drivers are provided with the Sun/CXP configuration of Doré. They are: the Rasterfile device driver, the SunView CXP device driver, and the Generic SunView device driver. Each of these device drivers is briefly described below. For more information on the device drivers see the appendix on Device Drivers.

Rasterfile Device Driver

The Rasterfile device driver is accessed from Doré applications by creating a device of type *"rasterfile"* in a call to *DoDevice*. For example:

```
device = DoDevice ("rasterfile", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Rasterfile device driver are:

–width *size*
–height *size*
–filename *name*

SunView CXP Device Driver

The SunView CXP device driver is accessed from Doré applications by creating a device of type *"sunviewGP"* in a call to *DoDevice*. For example:

```
device = DoDevice ("sunviewGP", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the SunView CXP device driver are:

–singlebuffered
–width *size*
–height *size*
–canvas *canvasid*

Generic SunView Device Driver

The Generic SunView device driver is accessed from Doré applications by creating a device of type *"sunview"* in a call to *DoDevice*. For example:

```
device = DoDevice ("sunview", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Generic SunView device driver are:

–width *size*
–height *size*
–canvas *canvasid*

Renderers

The Sun/CXP version of Doré includes two renderers. They are the Dynamic Renderer and the Stardent Production Renderer. The Dynamic Renderer is installed so that it is used when the render style (*DvSetRendStyle*) is *DcRealTime*. The Stardent Production Renderer is installed so that it is used when the render style is *DcProductionTime*.

For more details on which Doré attributes and primitives are used by particular renderers see the appendix on Renderers. The Dynamic Renderer depends heavily on the particular device driver being used, for more information about the capabilities of the device drivers see the appendix on Device Drivers.

Floating Point Precision

Doré allows both a single precision and a double precision application interface for Doré real numbers (*DtReal*). A library can be built from Portable Doré source to support either precision for Doré real numbers. By convention the single precision library is *libdore.a* and the double precision library is *libdore_d.a*. The include file *dore.h* allows compile time selection of the floating point precision.

To use single precision, files should be compiled with the *-DDORE_REAL_SINGLE*, and linked with *libdore.a*, for example:

```
cc -DDORE_REAL_SINGLE test.c -o test -ldore -lsuntool \
        -lsunwindow -lpixrect -lm
```

To use double precision, files should be compiled with the *-DDORE_REAL_DOUBLE*, and linked with *libdore_d.a* for example:

```
cc -DDORE_REAL_DOUBLE test.c -o test -ldore_d -lsuntool \
        -lsunwindow -lpixrect -lm
```

DORÉ DEVICE DRIVERS

APPENDIX F

Contents of this Appendix

This appendix contains the following subsections:

DORÉ DEVICE DRIVER OVERVIEW

APPENDIX F1

Overview

This section describes the components of a Doré device driver. A Doré device driver links the Doré kernel and the renderers with the devices on which output is displayed. Each device driver is composed of one required module, called the Device Control Module, and several Output Modules.

Doré provides well-defined interfaces to the modules so that renderers can access all device drivers in a similar manner. A renderer typically uses the Device Control Module functions and one of the Output Module Interfaces to display output. A renderer can therefore work with any device driver that supports its required Output Module Interface. Each time a renderer performs an update it is passed the handle of the Doré device object to be used to display its output, thus the user can dynamically change the device driver used to display the renderer's results.

The Device Control Module interface provides access to standard device features, such as color map control and double buffer swapping. The Output Module interfaces provides access to a broader range of features. The Production Renderer Output Module interface provides pixel output, while the Dynamic Renderer Output Module interface provides high level access to a generalized graphics pipeline.

Device Control Module

Using the Device Control Module as a link, the device driver provides information to the Doré kernel concerning the characteristics of the device. The device driver also performs standard device functions, such as setting the color map. Device Control Module functions are shown in Table F1-1. All device drivers are required to support these functions.

Table F1-1. Device Control Module Functions

become current driver	inquire resolution
clear rectangle depth	inquire stereo
clear rectangle depth and color	inquire visual type
close device	set background color
create local data	set color entries
flush	set current view
get color entries	set depth buffer enable
initialize device	set depth buffer write
inquire pixel data	set foreground color
inquire auto size	set shade index
inquire clip list	swap buffers
inquire device extent	update geometry
inquire number of colors	write scanline byte

Device Output Modules

Depending on the capabilities of the underlying device, a device driver will support one or more of the Output Module Interfaces. Currently two Output Module Interfaces are defined: the Production Renderer Output Module Interface and the Dynamic Renderer Output Module Interface.

Production Renderer Output Module Interface

The Production Renderer Output Module Interface is a very simple interface, and is intended for renderers which generate pixel data. These renderers, typically ones that generate an image through complex software algorithms, require minimal output support from the device. Thus, the Production Renderer Output Module Interface consists of just one function.

Table F1-2. The Production Renderer Output Module Function

write rectangle byte rgb

Dynamic Renderer Output Module Interface

The Dynamic Renderer Output Module Interface, in contrast to the Production Renderer Output Module Interface, provides high-level device access. It expects the device to perform the complex tasks associated with a graphics pipeline, including transformations, clipping and shading. Table F1-3 lists the functions in

the Dynamic Renderer Output Module Interface.

Table F1-3. Dynamic Renderer Output Module Functions

apply clip volume
create local device data
create local view data
create local window data
delete local device data
delete local view data
delete local window data
get lcstowcsmat
get wcstofcsmat
initialize camera
initialize light
initialize studio environment
pop clip volume
pop lcstofcsmat
post initialization
pre initialization
push clip volume
push lcstofcsmat
render polyline
render display group
render line list
render point list
render triangle list
render triangle mesh
set ambient intensity
set ambient switch
set backface cullable
set backface cull switch
set clip switch
set clip volume
set depthcue
set depthcue switch
set diffuse color
set diffuse intensity
set diffuse switch
set hidden surface switch
set interpolation type
set light attenuation
set light color
set light intensity
set light spread angle
set light spead exponent
set light switch
set light type
set line type
set line width
set map bump
set map bump switch
set map diffuse color
set map diffuse color switch
set map environ
set map environ switch
set map transp intensity
set map transp intensity switch
set reflection switch
set representation type
set shadow switch
set shade index
set specular color
set specular factor
set specular intensity
set specular switch
set surface shader
set stereo
set stereo switch
set transp color
set transp intensity
set transp switch
set camera matrix
set parallel matrix
set perspective matrix
set projection matrix
start update
transform clip z point
update lcstowcsmat lokatfrm
update lcstowcsmat pop
update lcstowcsmat push
update lcstowcsmat rotate
update lcstowcsmat scale
update lcstowcsmat shear
update lcstowcsmat tfmmat
update lcstowcsmat transl
update local data

The pipeline functions are implemented in either software, hardware or some combination of both, depending on the underlying capabilities of the device. The interface requires that the device provide all functions in the interface, but does not require the implementation of all function attributes. Thus, the output module must accept the attributes, but is not required to use them. Separate subsections of the device driver appendix describe which attributes in the Dynamic Renderer Output Module Interface are supported by specific devices.

Installation for Portable Doré

Each Doré device driver provides an external function that can be used to install it in the Doré system. Typically, this function is called during Doré initialization as a part of the *DsInitializeSystem* procedure. The port specific routine *S_initialize_drivers*, called by *DsInitializeSystem* will call the appropriate installation function if the driver is a standard part of the port.

If a device driver is not a standard part of a port, then it can be installed by calling the installation routine immediately after the *DsInitializeSystem* call. Doré application programs using a device with a non-standard driver must also be linked with an additional object module or library containing the device driver.

RASTERFILE DEVICE DRIVER

APPENDIX F2

Overview

The Rasterfile device driver writes a generated Doré image to a file. The file is written in the format described in *Appendix C, Raster File and Memory Formats,* with the *pixel* attribute set to *r8g8b8*. The Rasterfile driver writes the image output from a renderer directly to a file. It is *not* related to *DoFileRaster* and *DoRaster* objects in the Doré system.

Using the Driver

The Rasterfile device driver is accessed from Doré applications by creating a device of type *"rasterfile"* in a call to *DoDevice*. For example:

```
device = DoDevice ("rasterfile", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Rasterfile device driver are:

-width *size*
: requests a raster file with the image width of *size*. The default width is 512.

-height *size*
: requests a raster file with the image height of *size*. The default height is 512.

-filename *name*
: requests that the raster file be named *name*. The default name is *test.img*.

Output Modules

The Rasterfile device driver supports only the Production Renderer Output Module interface.

Installation for Portable Doré

The external function for installing the Rasterfile device driver is:

```
Pr_rasterfile_install_driver(name)
   DtPtr name;
```

If the string pointer *name* is *DcNullPtr* then the default name for the device driver is used, otherwise the name *name* is used to identify this device driver.

If the Rasterfile device driver is not part of the standard port, then an additional object module or library containing the device driver must be linked into the application and the driver installed after the *DsInitializeSystem* call. For example, the *main( )* program would include:

```
 ...
DsInitializeSystem (0);
Pr_rasterfile_install_driver (DcNullPtr);
 ...
```

and the application would be built with:

```
cc -o application application.o rasterfile.o \
        -ldore [ other libraries ]
```

where *rasterfile.o* contains the Rasterfile device driver.

The Rasterfile device driver is included as a standard part of most ports, and can consequently be used by applications without first calling the initialization routine.

STANDARD X11 DEVICE DRIVER

APPENDIX F3

Overview

The Standard X11 device driver provides access to any graphics device that supports the X11 Protocol. This includes workstations running X windows and devices like X terminals.

Using the Driver

The Standard X11 device driver is accessed from Doré applications by creating a device of type *"stdx11"* in a call to *DoDevice*. For example:

```
device = DoDevice ("stdx11", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Standard X11 device driver are:

-singlebuffered
: requests a single buffered window. The default is a double buffered window.

-geometry *geomstring*
: requests a window with a particular position and size. The format for *geomstring* is "WxH+X+Y" or "WxH" (where W, H, X, and Y are the integer values for width, height, and the X, Y position of the upper left corner of the window). If the "WxH" format is used, X and Y are assumed to be zero.

-visualtype *vtype*
: requests a window with the given visual type. The possible values for *vtype* are *DcPseudoColor* for an 8–bit pseudocolor window or *DcStaticGrey* for a black and white window.

-window *xwindow*
: requests that the X window handle *xwindow* that the

application has opened be used for drawing. When using this option, the *–display* option should also be specified. The *–singlebuffered*, *–geometry*, and *–visualtype* options are ignored when a window is specified.

-display *xdisplay*
requests that the X display handle *xdisplay* that the application has opened be used as the X display device.

-zbuffer
requests that the device driver use a software Z buffer. The use of this Z buffer is then controlled by the *DoHiddenSurfSwitch* primitive attribute. Note, when the Z buffer is being used (*DoHiddenSurfSwitch* is set to *DcOn*) there is a significant reduction in rendering speed. Even if *DoHiddenSurfSwitch* is set to *DcOff* there are memory and processing requirements which reduce program speed.

Output Modules

The Standard X11 device driver supports both the Production Renderer Output Module Interface and the Dynamic Renderer Output Module Interface.

The Production Renderer Output Module Interface is very simple, and all functions are supported.

The Standard X11 device driver driver uses many of the Doré attributes that are passed to the Dynamic Renderer Output Module. Table F3-1 lists the attributes used.

Table F3-1. Dynamic Renderer Attributes

Attribute	Used
DoAmbientIntens	yes
DoAmbientSwitch	yes
DoBackfaceCullable	yes
DoBackfaceCullSwitch	yes
DoClipSwitch	
DoClipVol	
DoDepthCue	
DoDepthCueSwitch	
DoDiffuseColor	yes
DoDiffuseIntens	yes
DoDiffuseSwitch	yes
DoHiddenSurfSwitch	yes(1)

Table F3-1. Dynamic Renderer Attributes (continued)

Attribute	Used
DoInterpType	
DcConstantShade	yes
DcVertexShade	yes
DcSurfaceShade	
DoLightAttn	
DoLightColor	yes
DoLightIntens	yes
DoLightSpreadAngles	
DoLightSpreadExp	
DoLightSwitch	
DoLightType	
DcLightAmbient	yes
DcLightInfinite	yes
DcLightPoint	
DcLightPointAttn	
DcLightSpot	
DcLightSpotAttn	
DoLineType	
DoLineWidth	
DoTextureMapBumpSwitch	
DoTextureMapBump	
DcStdBumpMap	
DoCallback	
DoTextureMapDiffuseColorSwitch	
DoTextureMapDiffuseColor	
DcStdTableLookup	
DcStd3dTableLookup	
DoCallback	
DoTextureMapEnvironSwitch	
DoTextureMapEnviron	
DcStdEnvironMap	
DoCallback	
DoTextureMapTranspIntensSwitch	
DoTextureMapTranspIntens	
DcStdTableLookup	
DcStd3dTableLookup	
DoCallback	
DoReflectionSwitch	

Table F3-1. Dynamic Renderer Attributes (continued)

Attribute	Used
DoRepType	
DcPoints	yes
DcWireframe	yes
DcSurface	yes
DcOutlines	yes
DoShadowSwitch	
DoShadeIndex	yes
DoSpecularColor	yes
DoSpecularFactor	
DoSpecularIntens	yes
DoSpecularSwitch	yes
DoSurfaceShade	
DcShaderConstant	yes
DcShaderLightSource	yes
DoStereo	
DoStereoSwitch	
DoTranspColor	
DoTranspIntens	
DoTranspSwitch	
DoCameraMatrix	yes
DoParallel	yes
DoPerspective	yes
DoProjection	yes
get wcstofcs matrix	yes
get lcstowcs matrix	yes
push lcstowcs matrix	yes
pop lcstowcs matrix	yes
push lcstofcs matrix	yes
pop lcstofcs matrix	yes
transform clip z point	yes
DoLookAtFrom	yes
DoRotate	yes
DoScale	yes
DoShear	yes
DoTransformMatrix	yes
DoTranslate	yes

(1) only works when the device driver is created with the *-zbuffer* option.

The Standard X11 device driver does not support texture mapping, so all texture attributes are ignored.

Installation of the Device Driver

The external function for installing the Standard X11 device driver is:

```
Pr_x11_std1_install_driver(name)
   DtPtr name;
```

If the string pointer *name* is *DcNullPtr* then the default name for the device driver is used, otherwise the name *name* is used to identify this device driver.

If the Standard X11 device driver is not part of the standard port, then an additional object module or library containing the device driver must be linked into the application and the driver installed after the *DsInitializeSystem* call. For example, the *main( )* program would include:

```
 ...
DsInitializeSystem (0);
Pr_x11_std1_install_driver (DcNullPtr);
 ...
```

and the application would be built with:

```
cc -o application application.o x11_std1.o \
        -ldore [ other libraries ]
```

where *x11_std1.o* contains the Standard X11 device driver.

STARDENT 1500/3000 DEVICE DRIVER

APPENDIX F4

Overview

The Stardent 1500/3000 device driver provides access to the high performance 3D graphics of the Stardent 1500/3000 graphics supercomputers. The Stardent 1500/3000, also known as the TITAN, manufactured by Stardent Computer, is a double buffered 24–bit color workstation that runs the X11+ windowing system.

Using the Driver

The Stardent 1500/3000 device driver is accessed from Doré applications by creating a device of type *"ardentx11"* or *"stardentx11"* in a call to *DoDevice*. For example:

```
device = DoDevice ("stardentx11", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Stardent 1500/3000 device driver are:

–singlebuffered
: requests a single buffered window. The default is a double buffered window.

–geometry *geomstring*
: requests a window with a particular position and size. The format for *geomstring* is "WxH+X+Y" or "WxH" (where W, H, X, and Y are the integer values for width, height, and the X, Y position of the upper left corner of the window). If the "WxH" format is used, X and Y are assumed to be zero.

–visualtype *vtype*
: requests a window with the given visual type. The possible values for *vtype* are *DcDirectColor* for a full 24–bit color window or *DcPseudoColor* for an 8–bit pseudocolor window.

–window *xwindow*
requests that the X window handle *xwindow* that the application has opened be used for drawing. When using this option, the *–display* option should also be specified. The *–singlebuffered*, *–geometry*, and *–visualtype* options are ignored when a window is specified.

–display *xdisplay*
requests that the X display handle *xdisplay* that the application has opened be used as the X display device.

–stereo
requests a stereo window.

Output Modules

The Stardent 1500/3000 device driver supports both the Production Renderer Output Module Interface and the Dynamic Renderer Output Module Interface.

The Production Renderer Output Module Interface is very simple, and all functions are supported.

The Stardent 1500/3000 device driver uses most, but not all, of the Doré attributes that are passed to the Dynamic Renderer Output Module. Table F4-1 shows the attributes used.

Table F4-1. Dynamic Renderer Attributes

Attribute	Used
DoAmbientIntens	yes
DoAmbientSwitch	yes
DoBackfaceCullable	yes
DoBackfaceCullSwitch	yes
DoClipSwitch	yes
DoClipVol	yes(1)
DoDepthCue	yes
DoDepthCueSwitch	yes
DoDiffuseColor	yes
DoDiffuseIntens	yes
DoDiffuseSwitch	always on
DoHiddenSurfSwitch	yes
DoInterpType	
DcConstantShade	yes
DcVertexShade	yes
DcSurfaceShade	

Table F4-1. Dynamic Renderer Attributes (continued)

Attribute	Used
DoLightAttn	yes(2)
DoLightColor	yes
DoLightIntens	yes
DoLightSpreadAngles	yes(2)
DoLightSpreadExp	yes(2)
DoLightSwitch	yes
DoLightType	
DcLightAmbient	yes
DcLightInfinite	yes
DcLightPoint	yes(2)
DcLightPointAttn	yes(2)
DcLightSpot	yes(2)
DcLightSpotAttn	yes(2)
DoLineType	
DoLineWidth	
DoTextureMapBumpSwitch	
DoTextureMapBump	
DcStdBumpMap	
DoCallback	
DoTextureMapDiffuseColorSwitch	
DoTextureMapDiffuseColor	
DcStdTableLookup	
DcStd3dTableLookup	
DoCallback	
DoTextureMapEnvironSwitch	
DoTextureMapEnviron	
DcStdEnvironMap	
DoCallback	
DoTextureMapTranspIntensSwitch	
DoTextureMapTranspIntens	
DcStdTableLookup	
DcStd3dTableLookup	
DoCallback	
DoReflectionSwitch	
DoRepType	
DcPoints	yes
DcWireframe	yes
DcSurface	yes
DcOutlines	yes

Table F4-1. Dynamic Renderer Attributes (continued)

Attribute	Used
DoShadowSwitch	
DoShadeIndex	yes
DoSpecularColor	yes
DoSpecularFactor	yes
DoSpecularIntens	yes
DoSpecularSwitch	yes
DoSurfaceShade	
DcShaderConstant	yes
DcShaderLightSource	yes
DoStereo	yes
DoStereoSwitch	yes
DoTranspColor	
DoTranspIntens	yes
DoTranspSwitch	yes
DoCameraMatrix	yes
DoParallel	yes
DoPerspective	yes
DoProjection	yes
get wcstofcs matrix	yes
get lcstowcs matrix	yes
push lcstowcs matrix	yes
pop lcstowcs matrix	yes
push lcstofcs matrix	yes
pop lcstofcs matrix	yes
transform clip z point	yes
DoLookAtFrom	yes
DoRotate	yes
DoScale	yes
DoShear	yes
DoTransformMatrix	yes
DoTranslate	yes

(1) The operator is ignored and is assumed to be *DcClipAnd*. Only 8 clipping planes are supported.

(2) Supported only on Stardent 3000.

The Stardent 1500/3000 device driver does not support texture mapping, so all texture attributes are ignored.

The external function for installing the Stardent 1500/3000 device driver is:

```
Pr_x11_DGL_install_driver(name)
   DtPtr name;
```

If the string pointer *name* is *DcNullPtr* then the default name for the device driver is used, otherwise the name *name* is used to identify this device driver.

If the Stardent 1500/3000 device driver is not part of the standard port, then an additional object module or library containing the device driver must be linked into the application and the driver installed after the *DsInitializeSystem* call. For example, the *main()* program would include:

```
 ...
DsInitializeSystem (0);
Pr_x11_DGL_install_driver (DcNullPtr);
 ...
```

and the application would be built with:

```
cc -o application application.o x11_DGL.o \
        -ldore [ other libraries ]
```

where *x11_DGL.o* contains the Stardent 1500/3000 device driver.

GENERIC SUNVIEW DEVICE DRIVER

APPENDIX F5

Overview

The Generic SunView device driver provides access to the 2–D graphics on the Sun Microsystems Sun/3 or Sun/4 workstations. The Sun/3 or Sun/4 is any color Sun workstation running the SunView windowing system. The Generic SunView device driver uses the SunView Graphics Interface.

Using the Driver

The Generic SunView device driver is accessed from Doré applications by creating a device of type *"sunview"* in a call to *DoDevice*. For example:

```
device = DoDevice ("sunview", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the Generic SunView device driver are:

-width *size*
: requests a window of width *size* pixels.

-height *size*
: requests a window of height *size* pixels.

-canvas *canvasid*
: requests that the SunView canvas opened by the application be used for drawing. The *–width* and *–height* and options are ignored when a canvas is specified.

Output Modules

The Generic SunView device driver supports only the Production Renderer Output Module Interface.

Installation of the Device Driver

The external function for installing the Generic SunView device driver is:

```
Pr_sunview_gen_install_driver(name)
   DtPtr name;
```

If the string pointer *name* is *DcNullPtr* then the default name for the device driver is used, otherwise the name *name* is used to identify this device driver.

If the Generic SunView device driver is not part of the standard port, then an additional object module or library containing the device driver must be linked into the application and the driver installed after the *DsInitializeSystem* call. For example, the main program would include:

```
 ...
DsInitializeSystem (0);
Pr_sunview_gen_install_driver (DcNullPtr);
 ...
```

and the application would be built with:

```
cc -o application application.o sunview_gen.o \
        -ldore [ other libraries ]
```

where *sunview_gen.o* contains the Generic SunView device driver.

SUNVIEW CXP DEVICE DRIVER

APPENDIX F6

Overview

The SunView CXP device driver provides access to the 3D graphics accelerators on Sun Microsystems Sun/3 and Sun/4-CXP workstations. The Sun/3-CXP and Sun/4-CXP are 8–bit color workstations with CG framebuffers and GP graphics accelerators, running the SunView windowing system. The SunView CXP device driver makes use of the Sun GPSI Graphics interface.

Using the Driver

The SunView CXP device driver is accessed from Doré applications by creating a device of type *"sunviewGP"* in a call to *DoDevice*. For example:

```
device = DoDevice ("sunviewGP", "");
```

The second parameter specifies optional permanent initialization values for the device. The options available for use with the SunView CXP device driver are:

-singlebuffered
: requests a single buffered window. The default is a double buffered window.

-width *size*
: requests a window of width *size* pixels.

-height *size*
: requests a window of height *size* pixels.

-canvas *canvasid*
: requests that the SunView canvas opened by the application be used for drawing. The *–singlebuffered*, *–width* and *–height* options are ignored when a canvas is specified.

Output Modules

The SunView CXP device driver supports both the Production Renderer Output Module Interface and the Dynamic Renderer Output Module Interface.

The Production Renderer Output Module Interface is very simple, and all functions are supported.

The SunView CXP device driver uses many of the Doré attributes that are passed to the Dynamic Renderer Output Module. Table F6-1 shows which of those attributes are used.

Table F6-1. Dynamic Renderer Attributes

Attribute	Used
DoAmbientIntens	yes
DoAmbientSwitch	yes
DoBackfaceCullable	yes
DoBackfaceCullSwitch	yes
DoClipSwitch	
DoClipVol	
DoDepthCue	
DoDepthCueSwitch	
DoDiffuseColor	yes
DoDiffuseIntens	yes
DoDiffuseSwitch	yes
DoHiddenSurfSwitch	yes
DoInterpType	
DcConstantShade	yes
DcVertexShade	yes
DcSurfaceShade	
DoLightAttn	
DoLightColor	yes
DoLightIntens	yes
DoLightSpreadAngles	
DoLightSpreadExp	
DoLightSwitch	
DoLightType	
DcLightAmbient	yes
DcLightInfinite	yes
DcLightPoint	
DcLightPointAttn	
DcLightSpot	
DcLightSpotAttn	

Table F6-1. Dynamic Renderer Attributes (continued)

Attribute	Used
DoLineType	
DoLineWidth	
DoTextureMapBumpSwitch	
DoTextureMapBump	
DcStdBumpMap	
DoCallback	
DoTextureMapDiffuseColorSwitch	
DoTextureMapDiffuseColor	
DcStdTableLookup	
DcStd3dTableLookup	
DoCallback	
DoTextureMapEnvironSwitch	
DoTextureMapEnviron	
DcStdEnvironMap	
DoCallback	
DoTextureMapTranspIntensSwitch	
DoTextureMapTranspIntens	
DcStdTableLookup	
DcStd3dTableLookup	
DoCallback	
DoReflectionSwitch	
DoRepType	
DcPoints	yes
DcWireframe	yes
DcSurface	yes
DcOutlines	yes
DoShadowSwitch	
DoShadeIndex	yes
DoSpecularColor	yes
DoSpecularFactor	
DoSpecularIntens	yes
DoSpecularSwitch	yes
DoSurfaceShade	
DcShaderConstant	yes
DcShaderLightSource	yes
DoStereo	
DoStereoSwitch	
DoTranspColor	

Table F6-1. Dynamic Renderer Attributes (continued)

Attribute	Used
DoTranspIntens	
DoTranspSwitch	
DoCameraMatrix	yes
DoParallel	yes
DoPerspective	yes
DoProjection	yes
get wcstofcs matrix	yes
get lcstowcs matrix	yes
push lcstowcs matrix	yes
pop lcstowcs matrix	yes
push lcstofcs matrix	yes
pop lcstofcs matrix	yes
transform clip z point	yes
DoLookAtFrom	yes
DoRotate	yes
DoScale	yes
DoShear	yes
DoTransformMatrix	yes
DoTranslate	yes

The SunView CXP device driver does not support texture mapping, so all texture attributes are ignored.

Installation of the Device Driver

The external function for installing the SunView CXP device driver is:

```
Pr_sunview_cxp_install_driver(name)
   DtPtr name;
```

If the string pointer *name* is *DcNullPtr* then the default name for the device driver is used, otherwise the name *name* is used to identify the device driver.

If the SunView CXP device driver is not part of the standard port, then an additional object module or library containing the device driver must be linked into the application and the driver installed after the *DsInitializeSystem* call. For example, the *main( )* program would include:

```
 ...
DsInitializeSystem (0);
Pr_sunview_cxp_install_driver (DcNullPtr);
 ...
```

and the application would be built with:

```
cc -o application application.o sunview_cxp.o \
        -ldore [ other libraries ]
```

where *sunview_cxp.o* contains the SunView CXP device driver.

INDEX

D