METAFILE HEADER

Creation Date 10/21/94
Mod Date 4/7/95

TYPE

Parent Hierarchy	3DMF
Binary: 3DMF 0x33444D46	**Ascii:** 3DMetafile

SIZE

16

PARENT OBJECTS

none

DATA FORMAT

```
Uns16         majorVersion
Uns16         minorVersion
MetafileFlags flags
FilePointer   tocLocation

• As of this release:

majorVersion = 0
minorVersion = 8
• The final release of the metafile will be:

majorVersion = 1
minorVersion = 0

• MetafileFlags bitfield is:
```

Binary	Text
0x00000000	Normal
0x00000001	Stream
0x00000002	Database

SUBOBJECTS

none

DESCRIPTION

• The **metafile header** is the first object to appear in any metafile.
• Metafile **versions** of 1.x are expected to maintain some degree of compatibility.
• **Flags** indicate to a general degree of how the file is structured or should be read.

A **database** file indicates that the metafile is a library, and all objects that are “shared” appear in the table of contents.
A **stream** file indicates that no references exist in the metafile, so that a parsing program may discard encountered data when it is through with it.

If the **toc location** (**Table of Contents** location) is NULL, the entire file must be parsed to find a **Table Of Contents**.

EXAMPLE

```
3DMetafile (
  1 0 # version
  Normal
  toc>
)
...
toc: TableOfContents (
  ...
)
```

● BEGIN GROUP

Creation Date 2/24/95
Mod Date 3/1/95

TYPE

Parent Hierarchy 3DMF

Binary: bgng 0x62676E67

Ascii: BeginGroup

SIZE

sizes of contained objects + (8 * number of child objects)

PARENT OBJECTS

special

DATA FORMAT

SUBOBJECTS

DESCRIPTION

The begin group object is used similarly to the container object, except it is used as the starting delimiter for a group. This allows a naive parser to traverse a metafile without special casing the many types of groups that appear in the metafile spec. It also allows for a single mechanism that is used to declare a group.

Please note that all objects of type "group" MUST be contained in a begin group, to allow them to be identified as starting a group.

EXAMPLE

```
BeginGroup ( DisplayGroup ( ) )
  Triangle ( 0 0 1 0 0 0 0 1 0 )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )

BeginGroup (
  OrderedDisplayGroup ( )
  DisplayGroupState ( DoNotDraw )
)
  Triangle ( 0 0 1 0 0 0 0 1 0 )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )

BeginGroup ( InfoGroup ( ) )
  CString ( “Copyright (c) 1995” )
EndGroup ( )
```

Container

Creation Date 10/27/94
Mod Date 1/24/95

Type

Parent Hierarchy 3DMF

Binary: cntr 0x636E7472

Ascii: Container

Size

sizes of contained objects + (8 * number of child objects)

Parent Objects

special

Data Format

SubObjects

special

Description

• Used to bind objects together to form a single object.
• Container objects always contain other objects.
• The first object in the container is called the “root” object, and sets the scope of the remaining objects in the container, called “subobjects.”
• In general, the “root” object instantiates the object with its default values, and subobjects append information to the original “root” object.
• There is one exception to these encapsulation rules, which is “group” objects. Although “group” objects contain a list of other objects, they are delimited with another 3DMF object, the end group object.

Example

```
Container (
  Box ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 1 )
  )
)
```

End Group

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** 3DMF	
	Binary: endg 0x656E6467	**Ascii:** EndGroup
Size	0	
Parent Objects	none	

No Data

• Groups should be arranged into non-overlapping pairs of BeginGroup (group type/data) and an “EndGroup” object.

• All groups must be arranged into DAGs. (no cycles are permitted)

SubObjects

none

Description

This object is used as a delimiter for all **group** objects.

Example

```
# Empty group
BeginGroup ( OrderedDisplayGroup ( ) )
EndGroup ( )

# Group containing 1 object
BeginGroup ( DisplayGroup ( ) )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )

# Inline group referenced elsewhere

REDColor:
BeginGroup (
  DisplayGroup ( )
  DisplayGroupState ( IsInline )
)
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 0 )
  )
EndGroup ( )

BeginGroup ( DisplayGroup ( ) )
  Reference ( 1 ) # REDColor
  Cone () # Cone is RED
EndGroup ()
toc: TableOfContents (
  nextTOC> -1 2 0 12
  1
  1 REDColor>
)
```

Reference

Creation Date 10/21/94
Mod Date 1/24/95

Type

Parent Hierarchy	3DMF	
Binary: rfrn	0x7266726E	**Ascii:** Reference

Size

4

Parent Objects

may be substituted for any Shared object

Data Format

```
Uns32 refID

• if refID = 0, must contain subobjects

• if refID > 0, a TOC must exist in current
metafile that contains refID's resolution

• This refID is resolved in the current metafile
unless a Storage subobject is found in the
Reference
```

SubObjects

1 Storage object (optional)

Description

The **reference** object is used to instantiate an object multiple times in a metafile.

It may be substituted anywhere in the metafile for another "**Shared**" object. Only **shared** objects may be referenced.

References are resolved in the **Table Of Contents**. If a "Storage" object is specified as a subobject, it is assumed that the **reference** is external to the current metafile, and should be resolved in that external storage's table of contents.

Example

```
Reference ( 23 ) # internal reference
...
toc: TableOfContents (
  nextTOC> 35 -1 0 12
  ...
  20 CarFrame>
  21 Axle>
  23 WheelOfCar>
  ...
)

Container ( # external reference
  Reference ( 23 )
  UnixPath ( "parts/car.eb" )
)
```

Table Of Contents

Creation Date 10/21/94
Mod Date 2/24/95

Type	**Parent Hierarchy** 3DMF
	Binary: toc 0x746F6320 — **Ascii:** TableOfContents
Size	28 + (tocEntrySize * nEntries)
Parent Objects	

Data Format

```
FilePointer nextTOC
Uns32       refSeed
Int32       typeSeed
Uns32       tocEntryType
Uns32       tocEntrySize
Uns32       nEntries
TOCEntry    tocEntries

• refSeed > 0
• typeSeed < 0
• tocEntryType = 0 or 1
• tocEntrySize = 12 or 16, based upon tocEntryType
• the TOCEntry structure  is:
  - tocEntryType 0, tocEntrySize 12 is:

    Uns32       refID
    FilePointer objLocation

  - tocEntryType 1, tocEntrySize 16 is:

    Uns32       refID
    FilePointer objLocation
    ObjectType  objType
```

SubObjects

none

Description

The **table of contents** provides a means of resolving **references** within a file. The “nextTOC” file pointer points to the next **table of contents** in the file, or is NULL if no other **table of contents** exists.

The reference seed indicates the next available reference id available for **reference** objects. It is an unsigned positive number that is incremented with each addtional reference in a file. It is always one more than the maximum reference seed in a file.

The type seed indicates the next available type ID available for **type** objects. It is a negative number that is decremented with each additional **type** in a file. It is always one less than the minimum type seed in a file.

The **tocEntryType** and **tocEntrySize** are a set of paired values which indicate the size and type of information stored in a tocEntry.

The **tocEntries** are sorted by reference ID, in increasing order, to allow fast searching of the table of contents.

Example

```
3DMetafile (
  1 0
  Normal
  toc>
)
box23:
Mesh (
  45 # nVertices
  ...
)
Reference ( 1 )
Arrows:
BeginGroup ( DisplayGroup ( ) )
  Cone ( )
  Scale ( 0.2 0.1 0.2)
  Cylinder ( )
EndGroup ( )
Reference ( 2 )
Reference ( 4 )
...
Type ( -1 “Joe’s Garage:RepairHistory” )
...

-1 ( “Jim” “Fixed lug nut” 0.23 0.2 1.2 )

toc:
TableOfContents (
  nextTOC>
  5  # refSeed
  -2 # typeSeed
  0 12 # tocEntry Type/Size
  3 # nEntries
  1 box23>
  2 Arrows>
  4 Geom34>
)
```

● TYPE

Creation Date 10/24/94
Mod Date 1/24/95

TYPE	**Parent Hierarchy** 3DMF	
	Binary: type 0x74797065	**Ascii:** Type
SIZE	4 + sizeof(String)	
PARENT OBJECTS		

DATA FORMAT

```
Int32   typeID
String  owner

• typeID < 0
• owner string
```

SUBOBJECTS

none

DESCRIPTION

A **type** definition is used to declare a custom data type. A **type** definition may appear anywhere in a file, however, the custom type must be encountered before the custom object of that type is encountered..

All custom types in the metafile are negative numbers, and the typeID field begins at -1 and is decremented for each additional type. Only 2147483648 (or 2^31) custom types are permitted in a single metafile.

The owner string is an ISO 9070 registered owner string. Owner strings are unique globally for each type of custom data.

In the binary and text metafile, the typeID is used as the object type later in the file.

EXAMPLE

```
Type (
  -1
  “Joe’s Garage:BoltData”
)

...

-1 (
   -2.3 34 # Stress (kPA/area)
)
```

FACE ATTRIBUTE SET LIST

Creation Date 10/21/94
Mod Date 3/15/95

TYPE

Parent Hierarchy Data, AttributeSetList

Binary: fasl 0x6661736C

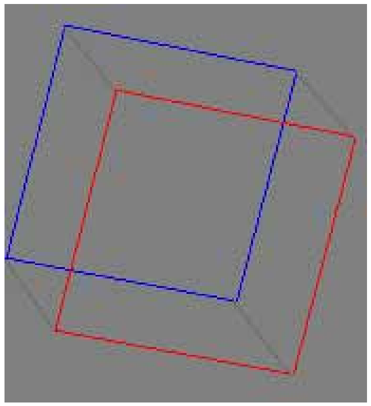
Ascii: FaceAttributeSetList

SIZE

12 + nIndices * sizeof(Uns) + padding

PARENT OBJECTS

ALWAYS: Box, GeneralPolygon, Mesh, TriGrid



DATA FORMAT

```
Uns32        nObjects
PackingEnum  packing
Uns32        nIndices
Uns32        indices[nIndices]
```

• nObjects must match parent values

• PackingEnum is:

Binary	Text
0x00000000	Include
0x00000001	Exclude

• 0 ≤ indices < nObjects

SUBOBJECTS

many AttributeSet (order-dependent)

DESCRIPTION

The **face attribute set list** specifies a list of attributes to be attached to a set of faces determined by the parent’s topology.

nObjects indicates the total number of objects being mapped to.

packing indicates how AttributeSet objects are mapped to indices. **Include** packing lists the face indices, in sequential order, of those faces to be assigned face attribute sets. **Exclude** packing lists the face indices, in sequential order, of those faces to NOT be assigned face attribute sets.

So, for example, supposing **nObjects** was 5, **Include** packing with a list of 3 indices after it means that there are 3 subobjects, each assigned to the indices in their order. **Exclude** packing with a list of 3 indices after it means there are 2 attribute sets subobjects, assigned to the indices NOT in the exclude list, in order.

The face attribute set list is padded to the nearest long word.

The values in **indices** always appear in increasing order.

If a packing value other than **Include** or **Exclude** is found, this object and its subobjects should be ignored.

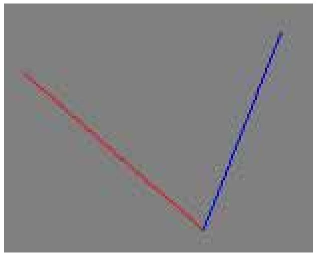
EXAMPLE

```
Container (
  Box ( )
  Container (
    FaceAttributeSetList (
      6 Include 2
      0 1
    )
    Container ( # assigned to 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # assigned to 1
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)

Container (
  Box ( )
  Container (
    FaceAttributeSetList (
      6 Exclude 2
      2 4
    )
    Container ( # assigned to 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # assigned to 1
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
    Container ( # assigned to 3
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
    Container ( # assigned to 5
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

Geometry Attribute Set List

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy**	Data, AttributeSetList
	Binary: gasl 0x6761736C	**Ascii:** GeometryAttributeSetList
Size	12 + nIndices * 4 + padding	
Parent Objects	ALWAYS: PolyLine	



Data Format

```
Uns32         nObjects
PackingEnum   packing
Uns32         nIndices
Uns32         indices[nIndices]
```

• nObjects must match parent values

• PackingEnum described in FaceAttributeSetList

SubObjects

many AttributeSet (order-dependent)

Description

The **geometry attribute set list** specifies a list of attributes to be attached to a set of geometric entities determined by the parent’s topology.

Currently, only the **PolyLine** primitive uses this object. Each **attribute set** is mapped to a line segment in the **PolyLine**.

Packing for this object is identical to the other attribute set lists.

Example

```
Container (
  PolyLine (
   3
   10 2 3
   0 0 0
   2 8.5 3
  )
  Container (
    GeometryAttributeSetList (
      3 Exclude 1 1
    )
    Container ( # segment 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # segment 2
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

VERTEX ATTRIBUTE SET LIST

Creation Date 10/21/94
Mod Date 3/15/95

TYPE

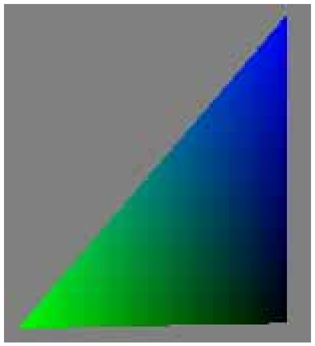
Parent Hierarchy	Data, AttributeSetList
Binary: vasl 0x7661736C	**Ascii:** VertexAttributeSetList

SIZE

12 + nIndices * sizeof(Uns) + padding

PARENT OBJECTS

ALWAYS: GeneralPolygon, Line, Mesh, Polygon, PolyLine, Triangle, TriGrid



DATA FORMAT

```
Uns32        nObjects
PackingEnum  packing
Uns32        nIndices
Uns32        indices[nIndices]
```

• nObjects must match parent values

• PackingEnum described in FaceAttributeSetList

SUBOBJECTS

many AttributeSet (order-dependent)

DESCRIPTION

The **vertex attribute set list** specifies a list of attributes to be attached to a set of vertices determined by the parent's topology.

Packing for this object is identical to the other attribute set lists.

EXAMPLE

```
Container (
  Triangle (
    0 0 0
    0 2 0
    0 0 2
  )
  Container (
    VertexAttributeSetList (
      3 Exclude 0
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 0 0 )
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
  )
)
```

CAMERA PLACEMENT

Creation Date
Mod Date 1/24/95

TYPE

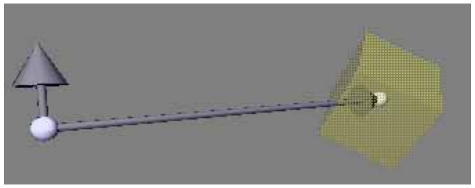
Parent Hierarchy	Data, CameraData
Binary: cmpl 0x636D706C	**Ascii:** CameraPlacement

SIZE

36

PARENT OBJECTS

ALWAYS: Camera objects: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera



DATA FORMAT

```
Point3D   location
Point3D   pointOfInterest
Vector3D  upVector
```

- upVector ⊥ (pointOfInterest - location)
- |upVector| = 1.0
- Default Values:

```
    0 0 1 # location
    0 0 0 # pointOfInterest
    0 1 0 # upVector
```

SUBOBJECTS

none

DESCRIPTION

The **camera placement** specifies the location and orientation of the camera in space, by a camera **location**, a **point of interest**, and an **up vector**. This placement locates and orients the camera, and defines a space in which the rest of the parameters are interpreted.

If the **up vector** is not of unit length upon reading, it should be normalized by the reading program.

The **camera placement** is affected by the current transformation state in a hierarchy. The **location** and **point of interest** are multiplied by the current transformation directly, and the **up vector** is multiplied by the current transformation minus any translation component of the transform, and unitized.

The **camera vector** is defined as:
camera vector = (**pointOfInterest** - **location**)

EXAMPLE

```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraPlacement (
     10 0 0 # located along X axis
      0 0 0 # point of interest is origin
      0 1 0 # Y is up
  )
)
```

Camera Range

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy	Data, CameraData
Binary: cmrg 0x636D7267	**Ascii:** CameraRange

Size

8

Parent Objects

ALWAYS: Camera objects: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera

Data Format

```
Float32   hither
Float32   yon
```

• 0 < hither ≤ yon

• default is:

```
hither  ε
yon     ∞
```

SubObjects

none

Description

The **camera range** affects the clipping of the viewing frustum.

This is used to bound the range of the set of objects of interest.

Hither is the frontmost clipping plane (sometimes referred to as “near”), **yon** is the backmost clipping plane (sometimes referred to as “far”).

Each of these distances is measured along the **camera vector**, described in the **Camera Placement** object.

Example

```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraRange (
     0.1 2 # hither, yon
  )
)
```

Camera Viewport

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy `Data, CameraData`

Binary: `cmvp` `0x636D7670`

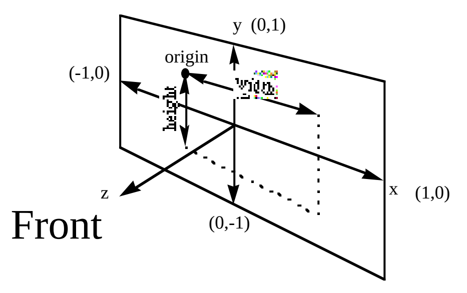
Ascii: `CameraViewPort`

Size

16

Parent Objects

ALWAYS: any Camera object: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera



Data Format

```
Point2D   origin
Float32   width
Float32   height
```

```
• -1 ≤ origin.x ≤ 1
• -1 ≤ origin.y ≤ 1
• 0 < width ≤ 2
• 0 < height ≤ 2
• Default is:
  -1 1  # origin
  2      # width
  2      # height
```

Description

The **camera viewport** specifies a rectangular region of the viewing frustum to which the image is clipped. Effectively the **view port** may be used to zoom in on a particular feature of an image.

The view port uses the cartesian coordinate system, with Y towards the top of the screen, X to the right, and Z coming towards the viewer, as shown in the diagram.

Example

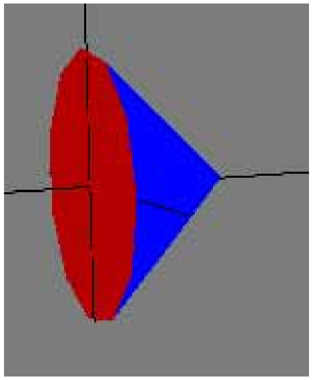
```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraViewPort ( # zoom to 200%
    -0.5 0.5 1 1
  )
)
```

SubObjects

none

Bottom Cap Attribute Set

Creation Date 10/21/94
Mod Date 4/7/95



Type

Parent Hierarchy Data, CapData

Binary: bcas 0x62636173

Ascii: BottomCapAttributeSet

Size

0

Parent Objects

ALWAYS: Cone, Cylinder

No Data

SubObjects

1 AttributeSet (optional)

Description

This object simply allows the attributes associated with the bottom cap of a **Cone** or **Cylinder** to be encapsulated.

Presence of a **bottom cap attribute set** does not neccessarily mean the bottom cap is drawn.

The **Caps** object determines whether the **Cone** and **Cylinder** caps are drawn or not.

Example

```
3DMetafile ( 1 0 Normal toc> )
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    capColor: Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
  )
)
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Reference (1)
  )
)
...
toc: TableOfContents (
   ...
   1 capColor>
)
```

CAPS

Creation Date 10/21/94
Mod Date 1/24/95

TYPE

Parent Hierarchy Data, CapData

Binary: caps 0x63617073

Ascii: Caps

SIZE

4

PARENT OBJECTS

ALWAYS: Cone, Cylinder

DATA FORMAT

```
CapsFlags caps

• CapsFlags is defined as:
```

Binary	Text
0x00000000	None
0x00000001	Bottom
0x00000002	Top

```
• Default is:
  None
```

SUBOBJECTS

none

DESCRIPTION

In the binary file, the upper 28 bits of the **caps** bitfield should be ignored. In the text file, unknown bitfield strings should be skipped. The default **caps** value is **0**, or **None**.

The **Top** cap bit (label) is ignored in the **Cone**.

EXAMPLE

```
Container (
  Cylinder ( )
  Caps ( Bottom | Top )
)

Container ( # Cone with a blue bottom
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

Face Cap Attribute Set

Creation Date 10/21/94
Mod Date 1/24/95

Type

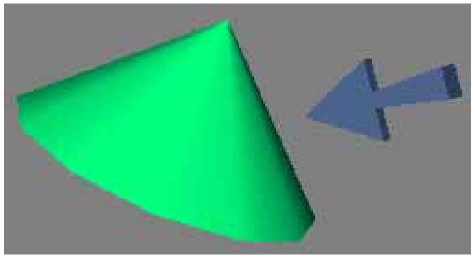
Parent Hierarchy	Data, CapData
Binary: fcas 0x66636173	**Ascii:** FaceCapAttributeSet

Size

0

Parent Objects

ALWAYS: Cone, Cylinder



No Data

SubObjects

1 AttributeSet (optional)

Description

Attaches a set of attributes to the face “cap” of the **cone** and **cylinder** primitives. For the cone, it’s indicated in the diagram.

Example

```
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0.2 0.9 0.4 )
    )
  )
)
```

Top Cap Attribute Set

Creation Date 10/21/94
Mod Date 4/4/95

Type

Parent Hierarchy	Data, CapData
Binary: tcas 0x74636173	**Ascii:** TopCapAttributeSet

Size

0

Parent Objects

ALWAYS: Cylinder

No Data

SubObjects

1 AttributeSet (optional)

Description

Attaches a set of attributes to the top “cap” of the **cylinder** primitive.

Presence of a **top cap attribute set** does not neccessarily mean the top cap is drawn.

The **Caps** object determines whether the **Cylinder** caps are drawn or not.

Example

```
Container (
  Cylinder ( )
  Caps ( Top )
  Container (
    TopCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0.2 0.9 0.4 )
    )
  )
)
```

DISPLAY GROUP STATE

Creation Date 10/21/94
Mod Date 4/7/95

TYPE

Parent Hierarchy Data

Binary: dgst 0x64677374

Ascii: DisplayGroupState

SIZE

4

PARENT OBJECTS

ALWAYS: DisplayGroup, OrderedDisplayGroup

DATA FORMAT

```
DisplayGroupStateFlags  traversalFlags

• DisplayGroupStateFlags is:
```

Binary	Text
0x00000000	None
0x00000001	Inline
0x00000002	DoNotDraw
0x00000004	NoBoundingBox
0x00000008	NoBoundingSphere
0x00000010	DoNotPick

• default is:

Binary	Text
0x00000000	None

SUBOBJECTS

none

DESCRIPTION

This piece of data is a subobject only to objects of type **display group**. It affects how a **display group** is traversed. These flags allow any **display group** to have the following characteristics:

• To have "invisible" objects in a scene which may act as user interface items, or may aid in bounding complex geometries
• To have non-user interface items which may serve only as decoration and should not be picked.
• To have a group of shaders/attributes which affects the state as an inline group so it may be instantiated and inherited in many parts of a hierarchy.

EXAMPLE

```
# to pick a chess piece by a box around it

BeginGroup ( DisplayGroup ( ) )
  PickIDStyle ( 1 )
  BeginGroup (
    DisplayGroup ( )
    DisplayGroupState ( DoNotDraw )
  )
    Scale ( 2 4 2 )
    Box ( )
  EndGroup ( )

  Container (
    DisplayGroup ( )
    DisplayGroupState ( DoNotPick )
  )
  Mesh ( # chess piece
    56 # nVertices
    0.2 0.3 0.5
    ...
  )
  EndGroup ( )
EndGroup ( )
```

General Polygon Hint

Creation Date 3/15/95
Mod Date 3/15/95

Type

Parent Hierarchy	Data
Binary: gplh 0x67706C68	**Ascii:** GeneralPolygonHint

Size

4

Parent Objects

ALWAYS: GeneralPolygon

Data Format

```
GeneralPolygonHintEnum   shapeHint

• GeneralPolygonHintEnum is:
```

Binary	Text
0x00000000	Complex
0x00000001	Concave
0x00000002	Convex

```
• default is:
    Complex
```

SubObjects

none

Description

The **GeneralPolygonHint** gives a reading application some hint of what shape a general polygon is.

A “Complex” general polygon may contain intersecting, concave, or convex polygons.

A “Concave” general polygon contains no intersecting polygons, but contains 1 or more concave polygons.

A “Convex” general polygon indicates that all contained polygons are convex and non-intersecting.

Example

```
Container (
  GeneralPolygon (
     1
     3
     0 2 3
     0 2 1
     2 0 0
  )
  GeneralPolygonHint ( Convex )
)
```

● Light Data

Creation Date 10/21/94
Mod Date 10/27/94

Type	**Parent Hierarchy** Data
	Binary: lght 0x6C676874 — **Ascii:** LightData
Size	20
Parent Objects	ALWAYS: any Light: SpotLight, AmbientLight, PointLight, DirectionalLight

Data Format

```
Boolean   isOn
Float32   intensity
ColorRGB  color

• 0 ≤ intensity ≤ 1

• Default is:
   True  # isOn
   1.0   # intensity
   1 1 1 # color
```

SubObjects

none

Description

The **light data** object affects information about a light that is common among all lights.

A **light** may be on or off, may vary in intensity, or may have different colors.

Example

```
Container (
  AmbientLight ( )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

Mesh Corners

Creation Date	10/21/94
Mod Date	3/15/95

Type

Parent Hierarchy Data

Binary: crnr 0x63726E72

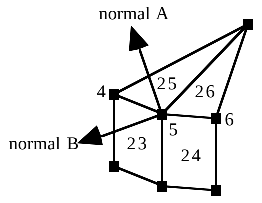
Ascii: MeshCorners

Size

```
4 + sizeof(corners[0..nCorners-1])

sizeof(MeshCorner) = 8 + nFaces * 4
```

Parent Objects

ALWAYS: Mesh



Data Format

```
Uns32       nCorners
MeshCorner  corners[nCorners]

• 0 < nCorners
• where MeshCorner is:

Uns32    vertexIndex
Uns32    nFaces
Uns32    faces[nFaces]

• 0 < nFaces
```

SubObjects

nCorners **AttributeSet**s (order-dependent)

Description

Mesh Corners allow you to attach **AttributeSet**s to a mesh vertex, to allow for attributes to be associated with a particular face-vertex pair. This may be used to allow sharp corners in an object (diagram above), to set different shading parameters for adjacent faces, etc.

Mesh corners supplies a vertex index, a list of face indices, and a **vertex attribute set** for each corner.

The **mesh corners** object most often appears inside a **container**, and always has **AttributeSet** subobjects. The first corner in the **mesh corners** data is mapped to the first **attribute set** subobject, the second corner to the second **attribute set**, etc.

Example

```
Container (
  Mesh (
    ...
  )
  Container (
    MeshCorners (
      2 # numCorners

      # Corner 0
      5 # vertexIndex
      2 # faces
      25 26 # face indices

      # Corner 1
      5 # vertexIndex
      2 # faces
      23 24 # face indices
    )
    Container (
      AttributeSet ( )
      Normal ( -0.2 0.8 0.3 )
    )
    Container (
      AttributeSet ( )
      Normal ( -0.7 -0.1 0.4 )
    )
  )
)
```

Mesh Edges

Creation Date 3/1/95
Mod Date 3/15/95

Type	**Parent Hierarchy** Data	
	Binary: edge 0x65646765	**Ascii:** MeshEdges
Size	4 + sizeof(corners[0..nCorners-1]) sizeof(MeshEdges) = 2 * sizeof(Uns)	
Parent Objects	ALWAYS: Mesh	

Data Format

```
Uns32       nEdges
MeshEdge    edges[nEdges]

• 0 < nEdges
• where MeshEdge is:

Uns32  vertexIndex1
Uns32  vertexIndex2
```

SubObjects

nCorners **AttributeSet**s (order-dependent)

Description

Mesh Edges allow you to attach **AttributeSet**s to a mesh edge.

You may attach mesh edges to any edge in the mesh that corresponds to a face edge. To specify and edge that should have an attribute set attached to it, include it as the nth edge the list of edges, and specify the attribute set as the nth attribute set subobject.

Example

```
Container (
  Mesh (
    ...
  )
  Container (
    MeshEdges (
      2 # numEdges
      0 1 # 1st edge vertexIndices
      1 2 # 2nd edge vertexIndices
    )
    Container ( /* 1st edge attribute set */
      AttributeSet ( )
      DiffuseColor ( 0.2 0.8 0.3 )
    )
    Container ( /* 2nd edge attribute set */
      AttributeSet ( )
      DiffuseColor ( 0.8 0.2 0.3 )
    )
  )
)
```

Nurb Curve 2D

Creation Date 10/21/94
Mod Date 1/24/95

Type	**Parent Hierarchy** Data	
	Binary: nb2c 0x6E623263	**Ascii:** NURBCurve2D
Size	8 + 12 * nPoints + 4 * (order + nPoints)	
Parent Objects	ALWAYS: TrimCurves	

Data Format

```
Uns32            order
Uns32            nPoints
RationalPoint3D  points[nPoints]
Float32          knots[order + nPoints]
```

```
• 2 ≤ order
• 2 ≤ nPoints
• 0 < points[...].w (weights of points)
```

SubObjects

none

Description

The **NURB Curve 2D** is a subobject of the **TrimCurves** object, and supplies a 2 dimensional curve to trim **NURB Patches**.

Example

Shader Data

Creation Date 10/21/94
Mod Date 3/22/95

Type

Parent Hierarchy Data

Binary: shdr 0x73686472

Ascii: ShaderData

Size

8

Parent Objects

ALWAYS: any Shader

Data Format

```
ShaderUVBoundaryEnum   uBounds
ShaderUVBoundaryEnum   vBounds
```

• ShaderUVBoundaryEnum is:

Binary	Text
0x00000000	Wrap
0x00000001	Clamp

• default is:
Wrap Wrap

SubObjects

none

Description

The **shader data** initializes boundary wrapping conditions for a **shader**.

Example

```
Container (
  CustomShader ( ... )
  ShaderData ( Wrap Clamp )
)
```

Shader Transform

Creation Date 10/21/94
Mod Date 4/7/95

Type

Parent Hierarchy	Data
Binary: sdxf 0x73647866	**Ascii:** ShaderTransform

Size

64

Parent Objects

ALWAYS: any Shader

Data Format

```
Matrix4x4   shaderTransform
```

SubObjects

none

Description

This transforms a shaded object into another world space coordinate system. It does not affect how the object is drawn, or the current state of the hierarchy.

Example

```
Container (
  3DMarbleShader ( )
  ShaderTransform (
    1 0 0 0
    0 1 0 0
    0 0 1 0
    2 3 4 1
  )
)
...
Type ( -3 “Apple:ATG:3DMarbleShader” )
Container (
  -3 ( 2.3 1.0 -10 )
  ShaderTransform (
    1 0 0 0
    0 1 0 0
    0 0 1 0
    2 3 4 1
  )
)
```

Shader UV Transform

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy Data		
Binary: sduv	0x73647576	**Ascii:** ShaderUVTransform

Size

36

Parent Objects

ALWAYS: any Shader

Data Format

```
Matrix3x3   matrix
```

SubObjects

none

Description

The **Shader UV transform** allows the uv's on a geometric object to be transformed before shading occurs.

This allows you to rotate a texture map, for example.

Example

```
Container (
  TextureShader ( )
  ShaderUVTransform (
    1 0 0
    0 1 0
    0.2 0.3 1
  )
  PixmapTexture (
    ...
  )
)
```

TRIM CURVES

Creation Date 10/21/94
Mod Date 4/7/95

TYPE

Parent Hierarchy	Data
Binary: trml 0x74726D63	**Ascii:** TrimLoop

SIZE

0

PARENT OBJECTS

ALWAYS: NURBPatch

NO DATA

SUBOBJECTS

many NURBCurve2D (order-dependent)

DESCRIPTION

The **Trim Loop** subobject allows users to attach trimming loops to a **NURB Patch**. The **Trim Loop** object contains no data, and serves only as an encapsulation of various 2-dimensional curves used for trimming.

The Trim loop object contains a sequence of 2 dimensional curves which are “concatenated” together to form a loop. The subobjects are order-dependent. Each trim loop subobject should contain loops that are geometrically continuous, meaning the first trim curve’s end point ends at the next trim curve’s starting point.

In the metafile version 1.0, the only 2-dimensional curve allowed is a **NURBCurve2D**.

In future releases of the metafile, we expect to add additional types of 2d trim curves for trimming NURBS.

EXAMPLE

```
Container (
 NURBPatch (
  4 4 4 4 # u,v order, num M,N points
  -2 2 0 1   -1 2 0 1   1 2 0 1   2 2 0 1
  -2 2 0 1   -1 2 0 1   1 0 5 1   2 2 0 1
  -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
  -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
   0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 # knots
 )
 Container (
  TrimLoop ( )
  NURBCurve2D (
    ...
  )
  NURBCurve2D (
    ...
  )
 )
)
```

● IMAGE CLEAR COLOR

Creation Date 1/20/95
Mod Date 1/24/95

◁ ▷

TYPE	**Parent Hierarchy** Data, ViewHintsData	
	Binary: imcc 0x696D6363	**Ascii:** ImageClearColor
SIZE	12	
PARENT OBJECTS	ALWAYS: ViewHints	

DATA FORMAT

```
ColorRGB  clearColor
```

SUBOBJECTS

none

DESCRIPTION

This specifies the preferred rgb color with should be used to clear the drawing area’s background.

EXAMPLE

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageClearColor ( 1 1 1 ) # white
)
Box ( )
```

IMAGE DIMENSIONS

Creation Date 1/20/95
Mod Date 1/24/95

TYPE	**Parent Hierarchy** Data, ViewHintsData	
	Binary: imdm 0x696646D	**Ascii:** ImageDimensions
SIZE	8	
PARENT OBJECTS	ALWAYS: ViewHints	

DATA FORMAT

```
Uns32   width
Uns32   height
```

- 0 < width
- 0 < height

SUBOBJECTS

none

DESCRIPTION

The **image dimensions** specifies the preferred image width and height in bits. It is a subobject of the **view hints**, which aids an application in determining how to display an image.

EXAMPLE

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageDimensions ( 32 32 )
  ImageClearColor ( 1 1 1 )
)
Rotate ( X 0.75 )
Rotate ( Y 0.75 )
Container (
  AttributeSet ( )
  DiffuseColor ( 1 0 0 )
)
Box ( )
```

Image Mask

Creation Date 1/20/95
Mod Date 3/23/95

Type

Parent Hierarchy	Data, ViewHintsData
Binary: immk 0x696D6D6D	**Ascii:** ImageMask

Size

12 + (rowBytes * height) + padding

Parent Objects

ALWAYS: ViewHints

Data Format

```
Uns32       width
Uns32       height
Uns32       rowBytes
EndianEnum  bitOrder
RawData     image[rowBytes * height]
```

• width, height in bits
• 0 < width
• 0 < height
• ((width >> 3) + ((width & 0x7) ? 1 : 0)) ≤ rowBytes
• EndianEnum is:

Binary	Text
0x00000000	BigEndian
0x00000001	LittleEndian

SubObjects

none

Description

The **image mask** is a bitmap that specifies how an image's rendered pixels should be clipped. The origin of the bitmap (the upper-left) is aligned with the origin (upper left) of the drawing area. Generally, the **image mask** and the **image dimensions** are used simultaneously to specify an image which is partially clipped.

The example to the right specifies a mask to clip a 32x32 image. The application using this data uses this clip mask to only render to a clipped portion of a custom document icon – in this case, the bitmap will only draw inside of a "document" icon, providing a small preview in the Finder with a black document icon. The image mask to the right was used to render the example above.

Example

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageDimensions ( 32 32 )
  ImageClearColor ( 1 1 1 )
  ImageMask (
    32 32 # width, height
    4 # rowBytes
    BigEndian # bitOrder
    0x000000000FFFF8000FFFF8000FFFF800
    0x0FFFF8000FFFF8000FFFF8000FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0FFFFFE00FFFFFE00FFFFFE00FFFFFE0
    0x0C61FFE00F24FFE00E64FFE00F24FFE0
    0x0F24FFE00C61FFE00FFFFFE000000000
  )
)
Rotate ( X 0.25 )
Rotate ( Y 0.23 )
Container (
  Torus ( 0 0.7 0 0 0 1 1 0 0 0 0 0 0.7 )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.9 0.9 )
  )
)
```

Ambient Coefficient

Creation Date 10/21/94
Mod Date 1/18/95

Type	**Parent Hierarchy** Element, Attribute	
	Binary: camb 0x63616D62	**Ascii:** AmbientCoefficient
Size	4	
Parent Objects	ALWAYS: AttributeSet	

Data Format

`Float32 ambientCoefficent`

• `0 ≤ ambientCoefficient ≤ 1.0`

SubObjects

none

Description

The **ambient coefficient** describes the intensity of the ambient light that is reflected by a surface.

Example

```
Container (
  AttributeSet ( )
  AmbientCoefficient ( 0.7 )
)
```

Diffuse Color

Creation Date 10/21/94
Mod Date 1/18/95

Type

Parent Hierarchy	Element, Attribute
Binary: kdif 0x6B646966	**Ascii:** DiffuseColor

Size

12

Parent Objects

ALWAYS: AttributeSet

Data Format

```
ColorRGB   diffuseColor
```

Description

The **diffuse color** indicates the amount of diffuse light reflected by a surface.

Example

```
Container (
  AttributeSet ( )
  DiffuseColor ( 1 0 0 ) # red
)
```

SubObjects

none

Highlight State

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** Element, Attribute	
	Binary: hlst 0x686C7374	**Ascii:** HighlightState
Size	4	
Parent Objects	ALWAYS: AttributeSet	

Data Format

```
Boolean  highlighted
```

SubObjects

none

Description

The **highlight state** attribute, when **true**, indicates that the current attribute state is overridden with the current **highlight style**'s **attribute set**. The **highlight state** attribute allows various portions of a **geometry** object to be highlighted for user interface, etc. while retaining the integrity of a **geometry**'s **attribute set**.

Example

```
Container (
  HighlightStyle ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 0 ) # RED
  )
)
...
Container (
  Polygon (
    3
    0 1 2
    0 0 0
    0 -1 2
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0 1 2 )
    HighlightState ( True )
    # polygon is drawn RED
  )
)
```

Normal

Creation Date 10/21/94
Mod Date 1/18/95

Type	**Parent Hierarchy**	Element, Attribute
	Binary: nrml 0x6E726D6C	**Ascii:** Normal
Size	12	
Parent Objects	ALWAYS: AttributeSet	

Data Format

```
Vector3D  normal
```

• |normal| = 1

SubObjects

none

Description

If **normal** is not of unit length upon reading, it should be normalized. (npi)

The **normal** indicates the surface normal at a vertex.

Example

```
Container (
  Polygon (
     5
     0.23423 0.56434 0.2312
     ...
  )
  Container (
    VertexAttributeSetList ( 5 Exclude 0 )
    Container (
      AttributeSet ( )
      Normal ( 0.8 -0.1 -0.1 )
    )
  )
)
```

● *Shading UV*

Creation Date 10/21/94
Mod Date 3/15/95

TYPE	**Parent Hierarchy** Element, Attribute	
	Binary: shuv 0x73687576	**Ascii:** ShadingUV
SIZE	8	
PARENT OBJECTS	ALWAYS: AttributeSet	

Data Format

```
Param2D  shadingUV

• Any UV parametrization is allowed, however,
shading generally occurs with the following values.

• 0 ≤ shadingUV.u ≤ 1
• 0 ≤ shadingUV.v ≤ 1
```

SubObjects

none

Description

The **shading UV** indicates an alternate UV to the **Surface UV** for shading purposes.

Shading UV's are generally used by **shaders** that affect appearance information, such as texture maps, which alter the color on a geometric surface.

Surface UV's are generally used for trimming.

Example

```
Container (
  AttributeSet ( )
  ShadingUV ( 0 0 )
)
```

Specular Color

Creation Date 10/21/94
Mod Date 3/15/95

TYPE	**Parent Hierarchy** Element, Attribute
	Binary: kspc 0x6B737063 — **Ascii:** SpecularColor
SIZE	12
PARENT OBJECTS	ALWAYS: AttributeSet

Data Format

```
ColorRGB   specularColor
```

SubObjects

none

Description

The **specular color** indicates the color of specular highlights on a surface.

Example

```
Container (
  AttributeSet ( )
  DiffuseColor ( 0.1 0.1 0.1 ) # near-black
  SpecularColor ( 1 1 1 ) # white
)
Sphere (
  0 0 0
  0 1 0
  1 0 0
  0 0 1
)
```

Specular Control

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** Element, Attribute	
	Binary: cspc 0x63737063	**Ascii:** SpecularControl
Size	4	
Parent Objects	ALWAYS: AttributeSet	

Data Format

```
Float32  specularControl

• 0 ≤ specularControl
```

SubObjects

none

Description

The **specular control** attribute indicates the power to which the specular component of lighting computations is raised.

Example

```
Container (
  AttributeSet ( )
  DiffuseColor ( 0.5 0.5 0.5 ) # near-black
  SpecularColor ( 0.5 ) # white highlights
  SpecularControl ( 1 ) # larger highlight area
)
Sphere ( )
```

Surface Tangent

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** Element, Attribute
	Binary: srtn 0x7372746E — **Ascii:** SurfaceTangent
Size	24
Parent Objects	ALWAYS: AttributeSet

Data Format

```
Vector3D  paramU
Vector3D  paramV
```

SubObjects

none

Description

The **surface tangent** attribute indicates the direction of changing U and V on a surface.

Example

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      ...
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0.1 0.293 )
      SurfaceTangent (
        1 0 0
        0 1 0
      )
    )
  )
)
```

Surface UV

Creation Date 10/21/94
Mod Date 3/15/95

TYPE	**Parent Hierarchy** Element, Attribute	
	Binary: sruv 0x73727576	**Ascii:** SurfaceUV
SIZE	8	
PARENT OBJECTS	ALWAYS: AttributeSet	

Data Format

```
Param2D  surfaceUV

• Any UV parametrization is allowed, however,
shading generally occurs with the following values.

• 0 ≤ surfaceUV ≤ 1
• 0 ≤ surfaceUV ≤ 1
```

SubObjects

none

Description

The **surface UV** indicates an alternate UV to the **shading UV** for shading purposes.

Surface UV’s are generally used for trim shaders.

Shading UV’s are generally used by shaders that affect appearance information, such as texture maps, which alter the color on a geometric surface.

Example

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      200 Include 4 10 21 22 11
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0 0 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0 1 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 1 1 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 1 0 )
    )
  )
)
```

Transparency Color

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy	Element, Attribute
Binary: kxpr 0x6B787072	**Ascii:** TransparencyColor

Size

12

Parent Objects

ALWAYS: AttributeSet

Data Format

```
ColorRGB  transparency
```

SubObjects

none

Description

The **transparency color** indicates the degree of light allowed to pass though the various channels (r,g,b) of a surface.

A color of (1, 1, 1) indicates complete transparency (meaning 100% of the light behind an object is allowed to pass through), a color of (0, 0, 0) indicates complete opacity (meaning no light passes through an object.)

Example

```
Container (
  Polygon (
    ...
  )
  Container (
    AttributeSet ( )
    TransparencyColor ( 1 0 0 )
  )
)
```

● Generic Renderer

Creation Date 10/21/94
Mod Date 2/24/95

◁ ▷

Type

Parent Hierarchy Shared, Renderer

Referencable

Binary: gnrr 0x676E7272

Ascii: GenericRenderer

Size

0

Parent Objects

SOMETIMES: ViewHints

No Data

SubObjects

none

Description

A renderer that doesn’t do anything, but may be used to accumulate state or for picking.

Example

```
Container (
  ViewHints ( )
  GenericRenderer ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

Interactive Renderer

Creation Date 10/21/94
Mod Date 4/6/95

TYPE	**Parent Hierarchy** Shared, Renderer — Referencable
	Binary: ctwn 0x6374776E — **Ascii:** InteractiveRenderer
SIZE	0
PARENT OBJECTS	SOMETIMES: ViewHints

No Data

SubObjects

none

Description

The **interactive** renderer.

This will be renamed later when the corresponding product is named.

Example

```
Container (
  ViewHints ( )
  InteractiveRenderer ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

Wire Frame Renderer

Creation Date 10/21/94
Mod Date 2/24/95

Type

Parent Hierarchy Shared, Renderer		Referencable
Binary: wrfr	0x77726672	**Ascii:** WireFrame

Size

0

Parent Objects

SOMETIMES: ViewHints

No Data

SubObjects

none

Description

A **wireframe** renderer.

Example

```
Container (
  ViewHints ( )
  Wireframe ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

Attribute Set

Creation Date 10/21/94
Mod Date 1/24/95

Type

Parent Hierarchy	Shared, Set	Referencable
Binary: attr 0x61747472	**Ascii:** AttributeSet	

Size

0

Parent Objects

any AttributeSetList, any Geometry, any Group, any CapAttributeSet

No Data

SubObjects

```
1 AmbientCoefficient (optional)
1 DiffuseColor (optional)
1 HighlightState (optional)
1 Normal (optional)
1 ShadingUV (optional)
1 SpecularColor (optional)
1 SpecularControl (optional)
1 SurfaceTangent (optional)
1 SurfaceUV (optional)
```

Description

A **attribute set** groups sets of unique attributes together and is associated with a vertex, face, or an entire geometry. Any object that is an **Element** may be placed in an **attribute set**.

An **attribute set** also may be placed in a group. The various attributes in an attribute set are inherited to nodes lower than it in a hierarchy.

Example

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      30 Exclude 2
      29 30
    )
    ...
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
      SurfaceUV ( 0.87 0.57 )
    )
    ...
  )
)
```

Orthographic Camera

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy Shared, Shape, Camera

Drawable Referencable

Binary: orth 0x6F727468

Ascii: OrthographicCamera

Size

16

Parent Objects

SOMETIMES: ViewHints

Data Format

```
Float32 left
Float32 top
Float32 right
Float32 bottom
```

- left < right
- bottom < top

SubObjects

```
1 CameraPlacement (optional, default)
1 CameraViewPort (optional, default)
1 CameraRange (optional, default)
```

Description

The lens characteristics are set with the dimensions of a rectangular view port in the frame of the camera.

Example

```
OrthographicCamera (
  -1 -1 1 1
)

Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraPlacement (
    0 0 20
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

View Angle Aspect Camera

Creation Date 10/21/94
Mod Date 1/24/95

Type

Parent Hierarchy	Shared, Shape, Camera	Drawable Referencable
Binary: vana 0x76616E61		**Ascii:** ViewAngleAspectCamera

Size

8

Parent Objects

SOMETIMES: ViewHints

Data Format

```
Float32 fieldOfView
Float32 aspectRatioXtoY

• 0 < fieldOfView ≤ π
• 0 < aspectRatioXtoY
```

SubObjects

```
1 CameraPlacement (optional, default)
1 CameraViewPort (optional, default)
1 CameraRange (optional, default)
```

Description

A perspective camera specified in terms of the minimum view angle and the aspect ratio of X to Y.

Example

```
ViewAngleAspectCamera (
  1.7 1.0
)

Container (
  ViewAngleAspectCamera (
    1.7 1.0
  )
  CameraPlacement (
    0 0 20
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

View Plane Camera

Creation Date 10/21/94
Mod Date 1/24/95

Type	**Parent Hierarchy** Shared, Shape, Camera	Drawable Referencable
	Binary: vwpl 0x7677706C	**Ascii:** ViewPlaneCamera
Size	20	
Parent Objects	SOMETIMES: ViewHints	

Data Format

```
Float32 viewPlane
Float32 halfWidthAtViewPlane
Float32 halfHeightAtViewPlane
Float32 centerXOnViewPlane
Float32 centerYOnViewPlane
```

- 0 < viewPlane
- 0 < halfWidthAtViewPlane
- 0 < halfHeightAtViewPlane
- centerXOnViewPlane, centerYOnViewPlane may be any value

SubObjects

1 CameraPlacement (optional, default)
1 CameraViewPort (optional, default)
1 CameraRange (optional, default)

Description

A **view plane camera** is a **view angle aspect camera** specified in terms of an arbitrary view plane. This is most useful when setting the camera to look at a particular object.

The viewPlane is set to distance from the camera to the object.

The halfWidth is set to half the width of the cross section of the object, and the halfHeight equal to the halfWidth divided by the aspect ratio of the viewPort.

This is the only perspective camera with specifications for off-axis viewing, which is desirable for scrolling.

Example

```
ViewPlaneCamera (
  ...
)

Container (
  ViewPlaneCamera (
    20
    15.0 15.0
    18 29
  )
  CameraPlacement (
    0 0 20
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

Box

Creation Date 10/21/94
Mod Date 1/20/95

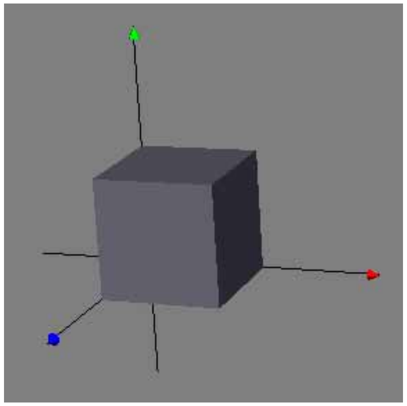
Type

Parent Hierarchy	Shared, Shape, Geometry — Drawable Referencable
Binary: box 0x626F7820	**Ascii:** Box

Size

0 or 48

Parent Objects



Data Format

```
Vector3D  orientation
Vector3D  majorAxis
Vector3D  minorAxis
Point3D   origin

• For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin
```

SubObjects

```
1 FaceAttributeSetList (optional, nObjects = 6)
1 AttributeSet (optional)
```

Description

This is a rectangular parallelepiped

A size of zero indicates the default values, helpful in instantiating a unit-cube.

The **Face Attribute Set List** subobject assigns color to the following faces:

Face $\perp$ orientation at origin + orientation

Face $\perp$ orientation at origin

Face $\perp$ majorAxis at origin + majorAxis

Face $\perp$ majorAxis at origin

Face $\perp$ minorAxis at origin + minorAxis

Face $\perp$ minorAxis at origin

Basically, the faces perpendicular to the “orientation” direction are assigned first, then the “majorAxis”, then the “minorAxis.”

Example

```
Box ( )

Box (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)

Container (
  Box ( )
  Container (
    FaceAttributeSetList (
      6 Exclude 0
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 1 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
```

Cone

Creation Date 10/21/94
Mod Date 1/14/95

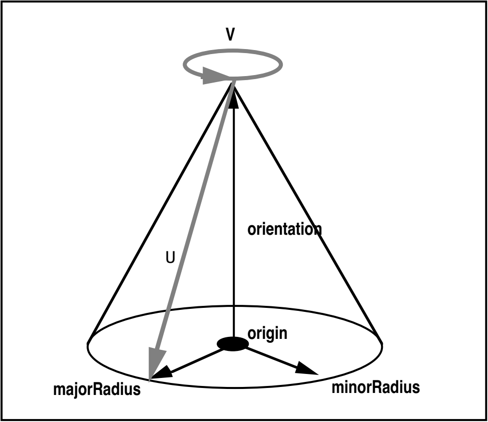
Type

Parent Hierarchy	Shared, Shape, Geometry	Drawable Referencable
Binary: cone	0x636F6E65	**Ascii:** Cone

Size

0 or 48

Parent Objects



Data Format

```
Vector3D  orientation
Vector3D  majorAxis
Vector3D  minorAxis
Point3D   origin

• For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin
```

SubObjects

```
1 Caps (optional, default)
1 FaceCapAttributeSet (optional)
1 BottomCapAttributeSet (optional)
1 AttributeSet (optional)
```

Description

A **cone** may have a cap, and may have attributes assigned to the entire **geometry**, to the “face” cap, or to the “bottom” cap.

The default parametrization is shown in the diagram.

Example

```
Cone ( )

Cone (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)

Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
```

Cylinder

Creation Date 10/21/94
Mod Date 1/14/95

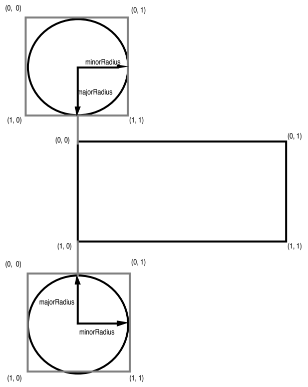
Type

Parent Hierarchy Shared, Shape, Geometry		Drawable Referencable
Binary: cyln	0x63796C6E	**Ascii:** Cylinder

Size

0 or 48

Parent Objects



Data Format

```
Vector3D  orientation
Vector3D  majorRadius
Vector3D  minorRadius
Point3D   origin

• For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin
```

SubObjects

```
1 Caps (optional, default)
1 TopCapAttributeSet (optional)
1 FaceCapAttributeSet (optional)
1 BottomCapAttributeSet (optional)
1 AttributeSet (optional)
```

Description

A **cylinder** may have either top or bottom caps, and may have attributes assigned to the entire geometry, to the “face” cap, the “bottom” cap, or the “top” cap.

The default parametrization is shown in the diagram.

Example

```
Cylinder ( )

Cylinder (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)

Container (
  Cylinder ( )
  Caps ( Bottom | Top )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
  )
  Container (
    TopCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
```

● *Disk*

Creation Date 10/21/94
Mod Date 1/14/95

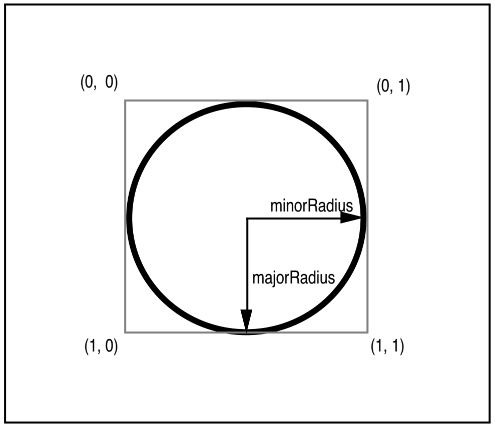
Type

Parent Hierarchy	Shared, Shape, Geometry — Drawable Referencable
Binary: disk 0x6469736B	**Ascii:** Disk

Size

0 or 36

Parent Objects



Data Format

```
Vector3D  majorRadius
Vector3D  minorRadius
Point3D   origin

• For 0-sized objects, default is:

1 0 0 # majorRadius
0 1 0 # minorRadius
0 0 0 # origin
```

SubObjects

```
1 AttributeSet (optional)
```

Description

This is an elliptical **disk** at the given origin with two vectors specifying the dimensions.

The default parametrization is shown in the diagram.

Example

```
Disk ( )

Disk (
  2 0 0
  0 1 1
  0 0 0
)

Container (
  Cylinder ( )
  Caps ( Bottom | Top )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

ELLIPSE

Creation Date 10/21/94
Mod Date 1/14/95

TYPE	**Parent Hierarchy** Shared, Shape, Geometry	Drawable Referencable
	Binary: elps 0x656C7073	**Ascii:** Ellipse
SIZE	0 or 36	
PARENT OBJECTS		

DATA FORMAT

```
Vector3D  majorAxis
Vector3D  minorAxis
Point3D   origin

• For 0-sized objects, default is:

1 0 0 # majorAxis
0 1 0 # minorAxis
0 0 0 # origin
```

SUBOBJECTS

```
1 AttributeSet (optional)
```

DESCRIPTION

This is an **ellipse** at the given origin with two vectors specifying its dimensions.

There is no default parametrization for an ellipse.

EXAMPLE

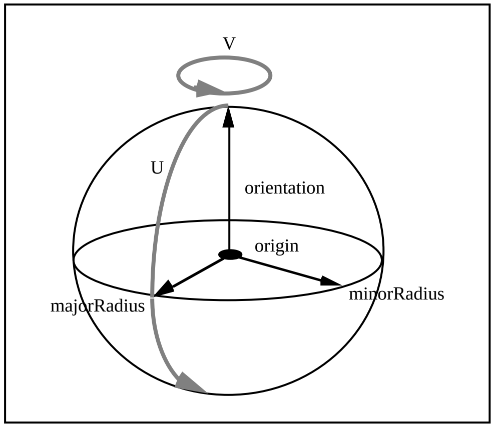
```
Ellipse ( )

Ellipse (
  2 0 0
  0 1 1
  0 0 0
)

Container (
  Ellipse ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

● ELLIPSOID

Creation Date 10/21/94
Mod Date 1/24/95

◁ ▷

TYPE	**Parent Hierarchy** Shared, Shape, Geometry — Drawable Referencable
	Binary: elpd 0x656C7064 — **Ascii:** Ellipsoid
SIZE	0 or 48
PARENT OBJECTS	



DATA FORMAT

```
Vector3D  orientation
Vector3D  majorRadius
Vector3D  minorRadius
Point3D   origin

• For 0-sized objects, default is:

1 0 0 # orientation
0 1 0 # majorRadius
0 0 1 # minorRadius
0 0 0 # origin
```

SUBOBJECTS

1 AttributeSet (optional)

DESCRIPTION

An **ellipsoid** may have an attribute set attached to it.

The default parametrization is shown in the diagram. V is zero to the left of majorRadius, and is 1 to the right. U is zero at the orientation vector, and 0 at the bottom.

EXAMPLE

```
Sphere ( )

Sphere (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
)

Container (
  Sphere ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

General Polygon

Creation Date 10/21/94
Mod Date 4/7/95

Type

Parent Hierarchy Shared, Shape, Geometry

Drawable Referencable

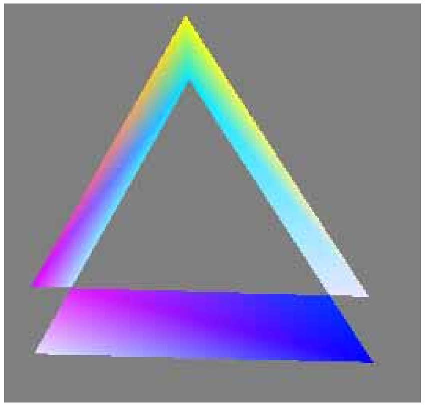
Binary: gpgn 0x6770676E

Ascii: GeneralPolygon

Size

```
4 + sizeof(polygons[0..nCoutours-1])

sizeof(PolygonData) = 4 + nVertices * 12
```

Parent Objects



Data Format

```
Uns32        nContours
PolygonData  polygons[nContours]

where PolygonData is:

Uns32   nVertices
Point3D vertices[nVertices]

• 0 < nContours
• 2 < nVertices
```

SubObjects

```
1 VertexAttributeSetList (optional, nObjects =
nVertices[0] + ... + nVertices[nContours-1])
1 AttributeSet (optional)
1 GeneralPolygonHint (optional)
```

Description

A **general polygon** is a polygon that may be convex or may contain holes. A general polygon also assumes that all faces are planar within floating point tolerances.

Holes are indicated by specifying a contour of the generalPolygon in clockwise order.

Polygons that cross use the even-odd rule to specify holes (see diagram).

You may specify the complexity of a GeneralPolygon by adding a viewHints object.

Example

```
Container (
 GeneralPolygon (
  2 # nContours
  3 # nVertices
  -1 0 0
  1 0 0
  0 1.7 0
  3 # nVertices
  -1 0.4 0
  1 0.4 0
  0 2.1 0
 )
 Container (
  VertexAttributeSetList ( 6 Exclude 2 0 4 )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 0 1 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 1 1 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 0 1 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 1 0 )
  )
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 1 1 1 )
 )
)
```

● LINE

Creation Date 10/21/94
Mod Date 1/14/95

TYPE

Parent Hierarchy Shared, Shape, Geometry   Drawable Referencable

Binary: line   0x6C696E65

Ascii: Line

SIZE

24

PARENT OBJECTS

DATA FORMAT

```
Point3D start
Point3D end
```

SUBOBJECTS

```
1 VertexAttributeSetList (optional, nObjects = 2)
1 AttributeSet (optional)
```

DESCRIPTION

Our basic **line** primitive is a line segment, a simple line drawn between two vertices.

Optional vertex attributes may be attached using a **VertexAttributeSetList**.

A set of attributes may be applied to the entire line segment by attaching an **attribute set**.

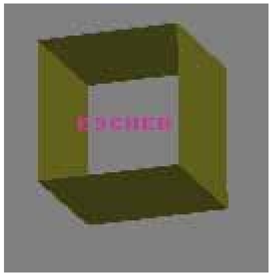
EXAMPLE

```
Line ( 0 0 0 1 0 0 )

Container (
  Line (
    0 0 0
    1 0 0
  )
  Container (
    VertexAttributeSetList ( 2 Exclude 0 )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

MARKER

Creation Date 10/21/94
Mod Date 3/23/95

TYPE	**Parent Hierarchy** Shared, Shape, Geometry — Drawable Referencable
	Binary: mrkr 0x6D726B72 — **Ascii:** Marker
SIZE	32 + (rowBytes * height) + padding
PARENT OBJECTS	



DATA FORMAT

```
Point3D    location
Int32      xOffset
Int32      yOffset
Uns32      width
Uns32      height
Uns32      rowBytes
EndianEnum bitEndian
RawData    data[height * rowBytes]
```

- 0 < width
- 0 < height
- (((width / 8) + ((width & 7) > 0)) ≤ rowBytes
- EndianEnum is:

Binary	Text
0x00000000	BigEndian
0x00000001	LittleEndian

SUBOBJECTS

1 AttributeSet (optional)

DESCRIPTION

The **marker** is used to rasterize bitmaps parallel to the viewing plane. They are used for annotation of an image.

EXAMPLE

```
Container (
 Marker (
  0.5 0.5 0.5 # origin
  -28 # xOffset
  -3 # yOffset
  56 # width
  6 # height
  7 # rowBytes
  BigEndian # bitOrder
  0x7E3C3C667E7C18606066666066187C3C
  0x607E7C661860066066607C1860066666
  0x6066007E3C3C667E6618
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 0.8 0.2 0.6 )
 )
)
```

Mesh

Creation Date 10/21/94
Mod Date 3/15/95

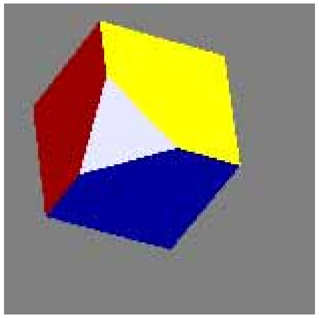
Type

Parent Hierarchy	Shared, Shape, Geometry	Drawable Referencable
Binary: mesh 0x6D657368	**Ascii:** Mesh	

Size

```
4 + nVertices * 12 +
  8 + (nFaces+nContours) * sizeof(faces[0..nFaces+nContours-1]

sizeof(MeshFace) = sizeof(Int) + sizeof(Uns) * nFaceVertexIndices
```

Parent Objects



Data Format

```
Uns32      nVertices
Point3D    vertices[nVertices]
Uns32      nFaces
Uns32      nContours
MeshFace   faces[nFaces + nContours]
```

• where MeshFace is:

```
Int32        nFaceVertexIndices
Uns32        faceVertexIndices[nFaceVertexIndices]
```

• 3 ≤ nVertices
• 3 ≤ nFaceVertexIndices

SubObjects

1 FaceAttributeSetList (optional, nObjects = nFaces)
1 VertexAttributeSetList (optional, nObjects = nVertices)
1 MeshCorners (optional)
1 AttributeSet (optional)

Description

The **mesh** is used for representing complex topological objects. It contains enough information to determine which polygonal faces are adjacent to each other without numerical ambiguity. This metafile object contains topological as well as geometrical information.

A contour (hole) in a face is indicated by supplying a negative number for the number of vertices, and adds a hole to the previous face that was not a contour.

The size of **nFaceVertexIndices** and **faceVertexIndices** is based on the value of **nVertices**.

We introduce a special subobject used only with the mesh, called "MeshCorners." This object allows multiple attribute sets to be attached to a single vertex, where each attribute set is bound to a set of vertex-face pairs. It can be used to place a sharp edge in the mesh (if the attribute set contains a normal, for instance).

Example

```
Mesh (
 10 # nVertices
 -1 1 1
 -1 1 -1
 1 1 -1
 1 -1 -1
 1 -1 1
 0 -1 1
 -1 -1 0
 -1 -1 -1
 1 1 1
 -1 0 1
 7 # nFaces
 0 # nContours
 3 6 5 9
 5 7 6 9 0 1
 4 2 3 7 1
 4 2 8 4 3
 4 1 0 8 2
 5 4 8 0 9 5
 5 3 4 5 6 7
)
```

Nurb Curve

Creation Date 10/21/94
Mod Date 1/10/95

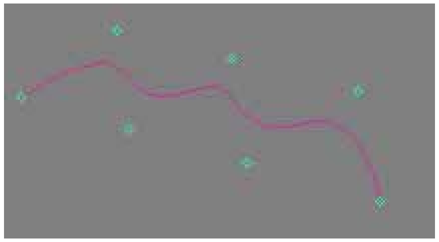
Type

Parent Hierarchy	Shared, Shape, Geometry — Drawable Referencable
Binary: nrbc 0x6E726263	**Ascii:** NURBCurve

Size

```
8 + (nPoints * 12) + ((nPoints + order) * 4)
```

Parent Objects



Data Format

```
Uns32            order
Uns32            nPoints
RationalPoint4D  points[nPoints]
Float32          knots[order + nPoints]
```

- 2 ≤ order
- 2 ≤ nPoints
- 0 < points[...].w (weights of points)

SubObjects

none

Description

NURB curves are Non-Uniform Rational B-spline curves. A rational B-spline curve is a curve in 4D space, which has been projected down to 3D space. Thus, the control points for a 3D rational curve have four components - x, y, z, and w (usually known as the weight). For such a point, the corresponding point in 3D space is (x/w, y/w, z/w)

Weights (w) are always positive.

Example

```
NURBCurve (
 4 7 # order, nPoints
 0 0 0 1 # points
 1 1 0 1
 2 0 0 1
 3 1 0 1
 4 0 0 1
 5 1 0 1
 6 0 0 1
 0 0 0 0 0.25 0.5 0.75 1 1 1 1 # knots
)
```

NURB PATCH

Creation Date 10/21/94
Mod Date 3/20/95

TYPE

Parent Hierarchy Shared, Shape, Geometry

Drawable Referencable

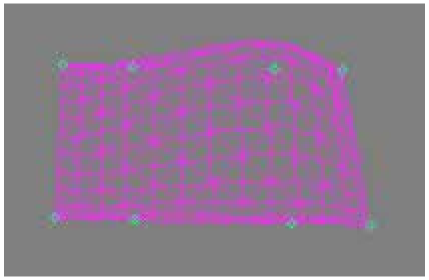
Binary: nrbp 0x6E726270

Ascii: NURBPatch

SIZE

```
16 + (16 * numColumns * numRows) + ((uOrder + numColumns) * 4) + ((vOrder +
numRows) * 4)
```

PARENT OBJECTS



DATA FORMAT

```
Uns32              uOrder
Uns32              vOrder
Uns32              numColumns
Uns32              numRows
RationalPoint4D    points[numMPoints*numNPoints]
Float32            uKnots[uOrder + numColumns]
Float32            vKnots[vOrder + numRows]
```

- 2 ≤ numColumns
- 2 ≤ numRows
- 2 ≤ uOrder
- 2 ≤ vOrder
- 0 < points[...].w (weights of points)

SUBOBJECTS

1 TrimCurves (optional)

DESCRIPTION

Non-Uniform Rational B-Spline (NURB) Patches are closed under projective transformations, can represent quadrics exactly, and can be refined locally to allow additional detail.

The default parametrization is given by the knot vectors.

Weights (w) are always positive.

EXAMPLE

```
NURBPatch (
 4 4 4 4 # u,v order, num M,N points
 -2 2 0 1   -1 2 0 1   1 2 0 1   2 2 0 1
 -2 2 0 1   -1 2 0 1   1 0 5 1   2 2 0 1
 -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
 -2 -2 0 1  -1 -2 0 1  1 -2 0 1  2 -2 0 1
  0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 # knots
)
```

● Point

Creation Date 10/21/94
Mod Date 1/24/95

Type	**Parent Hierarchy** Shared, Shape, Geometry	Drawable Referencable
	Binary: pnt 0x706E7420	**Ascii:** Point
Size	12	
Parent Objects		

Data Format

```
Point3D   point
```

SubObjects

```
1 AttributeSet (optional)
```

Description

The basic point primitive is an infinitesimally small point in space. It is specified as a 3D point plus an optional attribute set.

A 3D point has no default parametrization.

Example

```
Point ( 0 1 2 )
```

● Polygon

Creation Date 10/21/94
Mod Date 1/14/95

Type

Parent Hierarchy Shared, Shape, Geometry

Drawable Referencable

Binary: plyg 0x706C7967

Ascii: Polygon

Size

4 + nVertices * 12

Parent Objects

Data Format

```
Uns32   nVertices
Point3D vertices[nVertices]
```

• 2 ≤ nVertices

SubObjects

1 VertexAttributeSetList (optional, nObjects = nVertices)
1 AttributeSet (optional)

Description

The **polygon** is convex with no holes. To describe concave polygons or polygons with holes, use the “**general polygon**” primitive.

The points that make up a **polygon**’s face are assumed to be planar within floating point tolerances.

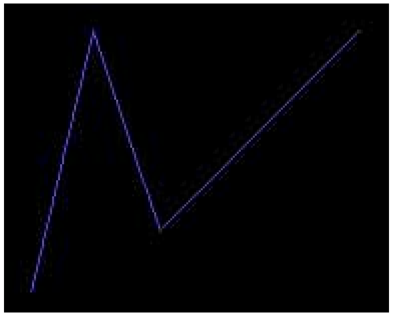
Example

```
Polygon (
  4
  0 1 1
  0 -1 1
  0 -1 -1
  0 1 -1
)
```

Poly Line

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** Shared, Shape, Geometry — Drawable Referencable
	Binary: plyl 0x706C796C — **Ascii:** PolyLine
Size	4 + nVertices * 12
Parent Objects	



Data Format

```
Uns32     nVertices
Point3D   vertices[nVertices]
```

• 2 ≤ nVertices

SubObjects

1 VertexAttributeSetList (optional, nObjects = nVertices)
1 GeometryAttributeSetList (optional, nObjects = nVertices - 1)
1 AttributeSet (optional)

Description

An extension of the basic line primitive is a **polyline**, where simple lines are drawn between adjacent points in a point list

A **polyline** is NOT closed, and the last point is never connected to the first point.

A **polyline** has no default parametrization.

Example

```
Container (
  PolyLine (
    4
    -1 -0.5 -0.25
    -0.5 1.5 0.45
    0 0 0
    1.5 1.5 1
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.4 0.2 0.9 )
  )
)
```

Torus

Creation Date 10/21/94
Mod Date 1/24/95

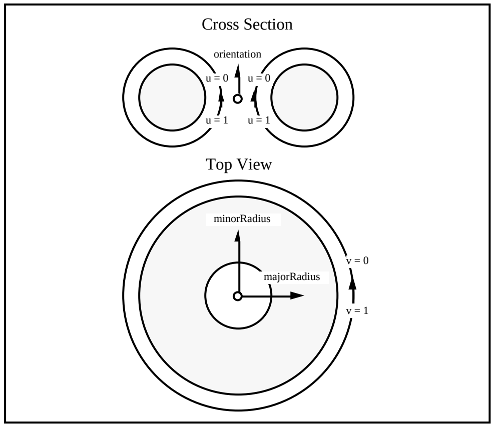
Type

Parent Hierarchy	Shared, Shape, Geometry — Drawable Referencable
Binary: tors 0x746F7273	**Ascii:** Torus

Size

0 or 52

Parent Objects



Data Format

```
Vector3D  orientation
Vector3D  majorAxis
Vector3D  minorAxis
Point3D   origin
Float32   ratio
```

• For 0-sized objects, default is:

```
1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin
1     # ratio
```

SubObjects

1 AttributeSet (optional)

Description

The orientation length specifies the radius of the circular along the orientation vector of the torus cross-section.

The major and minor axes are vectors to the center of the torus cross-section (as in the diagram).

The ratio is the change in the orientation length in the axial direction. A ratio of 2, for example, creates a fatter torus cross-section along the major and minor axes, a ratio of 0.5 creates a fatter cross-section along the orientation.

As far as anyone knows, the torus is useful for drawing donuts and bagels, and makes a great demo.

The default parametrization is shown in the diagram.

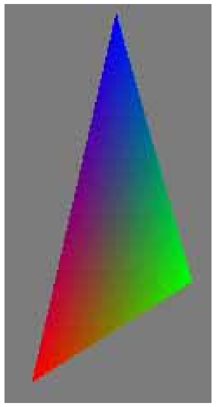
Example

```
Torus ( )

Torus (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
  1
)

Container (
  Torus ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

Triangle

Creation Date 10/21/94
Mod Date 1/14/95

TYPE	**Parent Hierarchy** Shared, Shape, Geometry	Drawable Referencable
	Binary: trng 0x74726E67	**Ascii:** Triangle
SIZE	36	
PARENT OBJECTS		



Data Format

```
Point3D  vertices[3]
```

SubObjects

```
1 VertexAttributeSetList (optional, nObjects = 3)
1 AttributeSet (optional)
```

Description

The most basic polygon is a **triangle**, which contains 3 points.

A **VertexAttributeSetList** may be used to attach attribute sets to the vertices (containing three vertex attribute sets) or an optional **AttributeSet** may be added to attach to the face.

There is no default parametrization for a triangle.

Example

```
Container (
 Triangle (
  -1 -0.5 -0.25
  0 0 0
  -0.5 1.5 0.45
 )
 Container (
  VertexAttributeSetList ( 3 Exclude 0 )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 0 0 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 1 0 )
  )
  Container (
   AttributeSet ( )
   DiffuseColor ( 0 0 1 )
  )
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 0.8 0.5 0.2 )
 )
)
```

Tri Grid

Creation Date 10/21/94
Mod Date 3/20/95

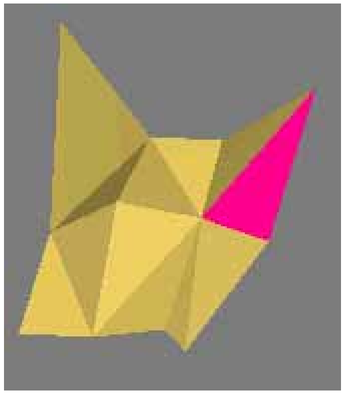
Type

Parent Hierarchy	Shared, Shape, Geometry	Drawable Referencable
Binary: trig 0x74726967		**Ascii:** TriGrid

Size

8 + (nColumns * nRows * 12)

Parent Objects



Data Format

```
Uns32     nColumns
Uns32     nRows
Point3D   points[numMVertices * numNVertices]
```

- 2 ≥ nColumns
- 2 ≥ nRows

SubObjects

1 FaceAttributeSetList (optional, nObjects = (numNVertices - 1) * (numMVertices - 1) * 2)
1 VertexAttributeSetList (optional, nObjects = numNVertices * numMVertices attribute sets)
1 AttributeSet (optional)

Description

Points specified are given in row major order.

You may add a **FaceAttributeSetList** to attach a set of attributes for each of the triangles generated by this primitive.

You may also add a **VertexAttributeSetList** to attach attributes to each vertex.

Example

```
Container (
 TriGrid (
  3 4 # nUVertices nVVertices
  -1 1 1      -0.5 1 0  0 1 0
   0.7 1 0.5  -1 0 0    -0.5 0 0.3
   0 0.2 0    0.5 0 0   -1 -1 0
   -0.5 -1 0  0 -1 0.1  0.2 -1.3 0.2
 )
 Container (
  FaceAttributeSetList ( 12 Include 1 5 )
  Container (
   AttributeSet ( )
   DiffuseColor ( 1 0 0.5 )
  )
 )
 Container (
  AttributeSet ( )
  DiffuseColor ( 0.8 0.7 0.3 )
 )
)
```

● GROUP

Creation Date 10/26/94
Mod Date 3/8/95

TYPE	**Parent Hierarchy** Shared, Shape — Drawable Referencable	
	Binary: grup 0x67727570	**Ascii:** Group
SIZE	0	
PARENT OBJECTS	none	

NO DATA

SUBOBJECTS

none

DESCRIPTION

The **group** is useful for grouping any type of shared objects together.

It is delimited by an **end group** object.

EXAMPLE

```
BeginGroup ( Group ( ) )
  CString ( “This is the first day of the rest
of your life.” )
  Torus ( )
EndGroup ( )
```

Display Group

Creation Date 10/21/94
Mod Date 1/24/95

Type	**Parent Hierarchy** Shared, Shape, Group — Drawable Referencable
	Binary: dspg 0x6C697374 — **Ascii:** DisplayGroup
Size	0
Parent Objects	

No Data

SubObjects

1 DisplayGroupState (optional)

Description

A **display group** contains only objects that are drawable.

A **display group** adds the ability to be traversed for various operations via the **DisplayGroupState** subobject.

It is delimited by an **end group** object.

Example

IO PROXY DISPLAY GROUP

Creation Date 1/24/95
Mod Date 4/6/95

TYPE

Parent Hierarchy Shared, Shape, Group, DisplayGroup

Drawable Referencable

Binary: iopx 0x70727879

Ascii: IOProxyDisplayGroup

SIZE

0

PARENT OBJECTS

NO DATA

SUBOBJECTS

1 DisplayGroupState (optional, default)

DESCRIPTION

The **IO proxy display group** contains drawable objects that are similar representations of the same object in different formats. For example, if it is known that a particular application does not understand NURBPatch's, the writing application may write the NURBPatch in an IO proxy group along with a mesh which is the tesselated NURBPatch.

The objects in a **IO proxy display group** appear in their preferencial order. The first object is the most preferred representation, the last object the least. The first object that is "understood" by a reading application should be used.

You may specify a group of objects inside a **IOProxyDisplayGroup**, as a group (up to its "**EndGroup**") delimiter is a single object.

It is understood that ONLY the first understood object in an **IO proxy display group** is traversed while drawing, bounding, or picking.

In other words, if an IO proxy display group contains many objects, only one of them will be drawn when it comes time to render an image, etc.

EXAMPLE

```
BeginGroup ( IOProxyDisplayGroup ( ) )
  Mesh (
    8
    0 0 0
    0 0 1
    0 1 0
    1 0 0
    1 1 0
    0 1 1
    1 0 1
    1 1 1
    ... etc.
  )
  Box ( )
EndGroup ( )

BeginGroup ( IOProxyDisplayGroup ( ) )
  NURBPatch (       # preferred object
    ...
  )
  DisplayGroup ( )  # 2nd choice object
    Translate ( 1 2 3 )
    Box ( )
  EndGroup ( )
EndGroup ( )
```

● *Ordered Display Group*

Creation Date 10/21/94
Mod Date 1/24/95

◁ ▷

Type

Parent Hierarchy Shared, Shape, Group, DisplayGroup

Drawable Referencable

Binary: ordg 0x6F72646C

Ascii: OrderedDisplayGroup

Size

0

Parent Objects

No Data

SubObjects

1 DisplayGroupState (optional, default)

Description

The **ordered display group** is simply a **display group** except that objects are sorted by type. Objects always appear in an **ordered group** in the following order:

• Transforms
• Styles
• AttributeSets
• Shaders
• Geometries
• DisplayGroups

It is delimited by an **end group** object.

Example

● Info Group

Creation Date 1/24/95
Mod Date 2/24/95

◁ ▷

Type

Parent Hierarchy `Shared, Shape, Group`   `Drawable` `Referencable`

Binary: `info` `0x696E666F`   **Ascii:** `InfoGroup`

Size

0

Parent Objects

none

No Data

SubObjects

none

Description

An **info group** contains nothing but **String** objects. It is used to add human-readable information pertaining to a file's origin or history. A use that comes to mind is copyright notices.

The **info group** object should be preserved by a reading application, and appended with additional information if a file is re-written.

It is delimited by an **end group** object.

Example

```
BeginGroup ( InfoGroup ( ) )
  CString (
    “Copyright © 1995 Apple Computer, Inc.” )
  CString (
    “Author: Bonanza Jellybean” )
EndGroup ( )
```

● Light Group

Creation Date 10/21/94
Mod Date 2/24/95

Type

Parent Hierarchy Shared, Shape, Group

Drawable Referencable

Binary: lghg 0x676C6768

Ascii: LightGroup

Size

0

Parent Objects

none

No Data

SubObjects

none

Description

A **light group** contains nothing but **lights**.

It is delimited by an **end group** object.

Example

```
BeginGroup ( LightGroup ( ) )
  AmbientLight ( )
  DirectionalLight ( 1 0 0 False )
EndGroup ( )
```

Ambient Light

Creation Date 10/21/94
Mod Date 10/27/94

Type

Parent Hierarchy Shared, Shape, Light

Drawable Referencable

Binary: ambn 0x616D626E

Ascii: AmbientLight

Size

0

Parent Objects

No Data

SubObjects

1 LightData (optional, default)

Description

An **ambient light** supplies light that comes from secondary reflections.

In lieu of other light sources, the **ambient light** illuminates the scene with a flat, uniform light.

Example

```
AmbientLight ( )

Container (
  AmbientLight ( )
  LightData (
    EcTrue # isOn
    1.0 # intensity
    1 0 0 # red color
  )
)
```

● Directional Light

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy Shared, Shape, Light

Drawable Referencable

Binary: drct 0x64726374

Ascii: DirectionalLight

Size

Parent Objects

Data Format

```
Vector3D    direction
Boolean     castsShadows
```

• |direction| = 1.0

SubObjects

1 LightData (optional, defaults)

Description

A **directional light** is far enough away from the scene that we may treat it as though it were infinitely far away. This produces shading results faster than any other type of light (except ambient).

It is specified with a vector pointing in the same direction as the light rays, an attenuation and a boolean value indicating whether this light casts shadows or not.

Example

```
DirectionalLight ( 1 0 0 True )

Container (
  DirectionalLight ( 1 0 0 True )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

● Point Light

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy Shared, Shape, Light

Drawable Referencable

Binary: pntl 0x706E746C

Ascii: PointLight

Size

Parent Objects

Data Format

```
Point3D      location
Attenuation  attenuation
Boolean      castsShadows
```

• where **Attenuation** is the structure:

```
Float32  c0
Float32  c1
Float32  c2
```

• **attenuation** is computed, using **d** as the distance from **location**:

$$\frac{1}{c0 + c1*d + c2 * d^2}$$

• $0 < c0$
• $0 < c1$
• $0 < c2$

• **attenuation** is not clamped to [0,1] to allow for lighting washout (such as in a nuclear explosion)

SubObjects

1 LightData (optional, defaults)

Description

A **point light** is a light at an infinitesimally small point in space. It may be attenuated or it may cast shadows.

Example

```
PointLight (
  12 23 2
  0 0 1 # InverseDistanceSquared
  True
)

Container (
  PointLight (
    12 23 2
    0 0 1 # InverseDistanceSquared
    True
  )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

Spot Light

Creation Date 10/21/94
Mod Date 1/24/95

Type

Parent Hierarchy Shared, Shape, Light — Drawable Referencable

Binary: spot 0x73706F74 **Ascii:** SpotLight

Size

Parent Objects

Data Format

```
Point3D       location
Vector3D      orientation
Boolean       castsShadows
Attenuation   attenuation
Float32       hotAngle
Float32       outerAngle
FallOffEnum   fallOff
```

• |orientation| = 1

• Attenuation is described in the Point Light

• 0 < hotAngle ≤ outerAngle ≤ π

• FallOffEnum is:

Binary	Text
0x00000000	None
0x00000001	Linear
0x00000002	Exponential
0x00000003	Cosine

SubObjects

1 LightData (optional, defaults)

Description

A **spot light** radiates with a circular cone of light that tapers toward the edge of the cone.

The hotSpotAngle is the angle (in radians) from the axis of the spot light for which the spot light has maximum, constant intensity. The outer angle is the angle for which the light falls to zero. Between these two, the light intensity tapers to zero according to the “FallOff” enumerated type.

Example

```
SpotLight (
  12 0 0
  0 1 0
  True
  0 0 1 # InverseDistanceSquared
  0.7 # hotAngle
  0.8 # outerAngle
  Cosine
)

Container (
  SpotLight (
    12 0 0
    0 1 0
    True
    0 0 1 # InverseDistanceSquared
    0.7 # hotAngle
    0.8 # outerAngle
   Cosine
  )
  LightData (
    True
    0.4
    1 0 1
  )
)
```

LAMBERT ILLUMINATION

Creation Date 10/21/94
Mod Date 10/27/94

TYPE

Parent Hierarchy Shared, Shape, Shader, IlluminationShader

Inherited
Drawable Referencable

Binary: lmil 0x6C6D696C

Ascii: LambertIllumination

SIZE

0

PARENT OBJECTS

NO DATA

SUBOBJECTS

none

DESCRIPTION

The lambertian illumination model.

EXAMPLE

```
LambertIllumination ( )
```

Phong Illumination

Creation Date 10/21/94
Mod Date 10/27/94

Type

Parent Hierarchy Shared, Shape, Shader, IlluminationShader

Inherited
Drawable Referencable

Binary: phil 0x7068696C

Ascii: PhongIllumination

Size

0

Parent Objects

No Data

SubObjects

Description

The phong illumination model.

Example

```
PhongIllumination ( )
```

TEXTURE SHADER

Creation Date 10/21/94
Mod Date 10/27/94

TYPE

Parent Hierarchy	Shared, Shape, Shader, SurfaceShader	Inherited: Drawable Referencable
Binary: txsu 0x74787375	**Ascii:** TextureShader	

SIZE

0

PARENT OBJECTS

NO DATA

SUBOBJECTS

1 PixmapTexture (required)

DESCRIPTION

The **texture shader** is used to perform shading using a texture (in this case, a PixmapTexture).

EXAMPLE

```
Container (
  TextureShader ( )
  PixmapTexture (
    ...
  )
)
```

● Backfacing Style

Creation Date 10/21/94
Mod Date 10/27/94

Type

Parent Hierarchy Shared, Shape, Style

Inherited
Drawable Referencable

Binary: bckf 0x62636B66

Ascii: BackfacingStyle

Size

Parent Objects

Data Format

```
BackfacingEnum  backfacing

• where BackfacingEnum is:

Text        Binary
0x00000000  Both
0x00000001  Culled
0x00000002  Flipped
```

SubObjects

none

Description

The **backfacing style** tells a renderer how to clip backfacing polygons while rendering.

Example

```
BackfacingStyle ( Culled )
```

● *FILL STYLE*

Creation Date 10/21/94
Mod Date 1/10/95

TYPE

Parent Hierarchy Shared, Shape, Style

Inherited
Drawable Referencable

Binary: fist 0x66697374

Ascii: FillStyle

SIZE

4

PARENT OBJECTS

DATA FORMAT

```
FillStyleEnum    fillStyle

• where FillStyleEnum is:
```

Text	Binary
0x00000000	Filled
0x00000001	Edges
0x00000002	Points
0x00000003	Empty

SUBOBJECTS

none

DESCRIPTION

The **fill style** tells a renderer what parts of a polygon to draw.

EXAMPLE

```
FillStyle ( Edges )
```

Highlight Style

Creation Date 10/21/94
Mod Date 3/8/95

Type	**Parent Hierarchy** Shared, Shape, Style	Inherited: Drawable Referencable
	Binary: high 0x68696768	**Ascii:** HighlightStyle
Size	0	
Parent Objects		

No Data

SubObjects

1 AttributeSet (required)

Description

The **highlight style** sets the binding for highlighting features of a geometry via the **HighlightState** attribute. The **attribute set** subobject sets the highlight attribute set.

Example

```
Container (
  HighlightStyle ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0 0 1 )
  )
)
```

Interpolation Style

Creation Date 10/21/94
Mod Date 10/27/94

Type

Parent Hierarchy Shared, Shape, Style

Inherited
Drawable Referencable

Binary: intp 0x696E7470

Ascii: InterpolationStyle

Size

4

Parent Objects

Data Format

```
InterpolationStyleEnum     interpolationStyle

• where InterpolationStyleEnum is:
```

Binary	Text
0x00000000	None
0x00000001	Vertex
0x00000002	Pixel

SubObjects

none

Description

The **interpolation style** tells a renderer how to interpolate shading values on a polygon.

Example

```
InterpolationStyle ( Vertex )
```

● ORIENTATION STYLE

Creation Date 10/21/94
Mod Date 3/8/95

◁ ▷

TYPE

Parent Hierarchy Shared, Shape, Style

Inherited
Drawable Referencable

Binary: ornt 0x6F726E74

Ascii: OrientationStyle

SIZE

4

PARENT OBJECTS

DATA FORMAT

```
OrientationEnum orientation

• where OrientationEnum is:

Binary      Text
0x00000000  CounterClockwise
0x00000001  Clockwise
```

DESCRIPTION

The Orientation style is used to change the orientation of polygons.

EXAMPLE

```
OrientationStyle ( Clockwise )
```

SUBOBJECTS

none

Pick Id Style

Creation Date 10/21/94
Mod Date 10/27/94

Type	**Parent Hierarchy** Shared, Shape, Style — Inherited Drawable Referencable
	Binary: pkid 0x706B6964 — **Ascii:** PickIDStyle
Size	4
Parent Objects	

Data Format

```
Uns32   id
```

SubObjects

none

Description

The **pick ID style** is used to allow the user to insert ids within a hierarchy to aid in picking a hierarchy.

Example

```
PickIDStyle ( 23 )
```

PICK PARTS STYLE

Creation Date 10/21/94
Mod Date 3/8/95

TYPE

Parent Hierarchy	`Shared, Shape, Style`	Inherited: Drawable Referencable
Binary: `pkpt` `0x706B7074`	**Ascii:** `PickPartsStyle`	

SIZE

PARENT OBJECTS

DATA FORMAT

```
PickPartsFlags    pickParts

• where PickPartsFlags is:

Text        Binary
0x00000000  Object
0x00000001  Face
0x00000002  Edge
0x00000004  Vertex

• default is:
  Object
```

SUBOBJECTS

```
none
```

DESCRIPTION

The **pick parts style** determines the level of granularity for picking.

EXAMPLE

```
PickPartsStyle ( Object | Vertex )
```

● Receive Shadows Style

Creation Date 10/21/94
Mod Date 10/27/94

Type	**Parent Hierarchy** Shared, Shape, Style	Inherited: Drawable Referencable
	Binary: rcsh 0x72637368	**Ascii:** ReceiveShadowsStyle
Size	4	
Parent Objects		

Data Format

```
Boolean  receiveShadows
```

SubObjects

none

Description

The **receive shadows style** determines whether a geometry receives shadows when rendering. It is coupled with the “casts shadows” field in all lights, excluding the ambient light.

Example

```
ReceiveShadowsStyle ( True )
```

Subdivision Style

Creation Date 10/21/94
Mod Date 3/15/95

Type

Parent Hierarchy	Shared, Shape, Style	Inherited: Drawable Referencable
Binary: sbdv 0x7364636C		**Ascii:** SubdivisionStyle

Size

```
(subdivisionMethod == Constant) ? 12 : 8
```

Parent Objects

Data Format

This object has two forms, based on the subdivison method field:

• for subdivisionMethod == WorldSpace or ScreenSpace the structure is:

```
SubdivisionMethodEnum subdivisionMethod
Float32               value1
```

• for subdivisionMethod == Constant, the values are integral:

```
SubdivisionMethodEnum subdivisionMethod
Uns32                 value1
Uns32                 value2
```

where SubdivisionMethodEnum is:

Binary	Text
0x00000000	Constant
0x00000001	WorldSpace
0x00000002	ScreenSpace

SubObjects

none

Description

The **subdivision style** tells a geometric decomposition the courseness of a geometric primitive tesselation. There are three methods of subdivision: constant, world space, and screen space subdivision.

Constant subdivision supplies 2 integral values, which indicate the number of sections the u and v axes of a decomposition should be divided into.

The Screen Space value indicates average size of a single polygon in a tesselation in screen space.

The world space value indicates the average size of a single polygon in a tesselation in world space.

Example

```
SubdivisionStyle (
   Constant 12 12
)

SubdivisionStyle (
   WorldSpace 50
)

SubdivisionStyle (
   ScreenSpace 50
)
```

Matrix Transform

Creation Date 10/21/94
Mod Date 10/27/94

Type	**Parent Hierarchy** Shared, Shape, Transform	Inherited: Drawable Referencable
	Binary: mtrx 0x6D747278	**Ascii:** Matrix
Size	64	
Parent Objects		

Data Format

```
Matrix4x4  matrix
```

• matrix is invertible

Description

A custom, invertible matrix transform.

Example

SubObjects

none

Quaternion Transform

Creation Date 10/21/94
Mod Date 1/24/95

Type	**Parent Hierarchy** Shared, Shape, Transform	Inherited: Drawable Referencable
	Binary: qtrn 0x7174726E	**Ascii:** Quaternion
Size	16	
Parent Objects		

Data Format

```
Float32   w
Float32   x
Float32   y
Float32   z
```

Description

The quaternion specifies three axes of rotation and a "twist" value.

Useful for user interface.

Example

```
Quaternion ( 0.2 0.7 0.2 1.57 )
```

SubObjects

none

ROTATE TRANSFORM

Creation Date 10/21/94
Mod Date 10/27/94

TYPE	**Parent Hierarchy** Shared, Shape, Transform		Inherited: Drawable Referencable
	Binary: rott 0x726F7474	**Ascii:** Rotate	
SIZE			
PARENT OBJECTS			

DATA FORMAT

```
AxisEnum    axis
Float32     radians

• AxisEnum is:

Binary        Text
0x00000000    X
0x00000001    Y
0x00000002    Z
```

SUBOBJECTS

none

DESCRIPTION

Rotate about the X, Y, or Z axes.

EXAMPLE

```
Rotate ( X 1.57 )
```

Rotate About Axis Transform

Creation Date 10/21/94
Mod Date 10/27/94

Type

Parent Hierarchy	Shared, Shape, Transform	Inherited: Drawable Referencable
Binary: rtaa 0x72746161	**Ascii:** RotateAboutAxis	

Size

28

Parent Objects

Data Format

```
Point3D  origin
Vector3D orientation
Float32  radians
```

• |orientation| = 1

SubObjects

none

Description

Rotate about an arbitrary axis in space.

Example

```
RotateAboutAxis (
  20 0 0 # origin
  0  1 0 # orientation
  1.57 # radians
)
```

ROTATE ABOUT POINT TRANSFORM

Creation Date 10/21/94
Mod Date 10/27/94

TYPE

Parent Hierarchy	Shared, Shape, Transform	Inherited Drawable	Referencable
Binary: rtap	0x72746170	**Ascii:** RotateAboutPoint	

SIZE

20

PARENT OBJECTS

DATA FORMAT

```
AxisEnum   axis
Float32    radians
Point3D    origin
```

• AxisEnum is:

Binary	Text
0x00000000	X
0x00000001	Y
0x00000002	Z

DESCRIPTION

To rotate about the X, Y, or Z axes at an arbitrary point in space.

EXAMPLE

SUBOBJECTS

none

● *Scale Transform*

Creation Date 10/21/94
Mod Date 10/27/94

Type	**Parent Hierarchy** Shared, Shape, Transform — Inherited: Drawable Referencable
	Binary: scal 0x7363616C — **Ascii:** Scale
Size	
Parent Objects	

Data Format

```
Vector3D   scale
```

Description

A scale transform.

Example

```
Scale ( 1 1 2 )
```

SubObjects

none

Translate Transform

Creation Date 10/21/94
Mod Date 10/27/94

Type

Parent Hierarchy	Shared, Shape, Transform	Inherited	Drawable Referencable
Binary: trns	0x74726E73	**Ascii:** Translate	

Size

12

Parent Objects

Data Format

```
Vector3D    translate
```

Description

A translate transfrom.

Example

```
Translate ( 1 2 100 )
```

SubObjects

none

Unknown Binary

Creation Date 4/6/95
Mod Date 4/7/95

Type	**Parent Hierarchy** Shared, Shape — Drawable Referencable
	Binary: ukbn 0x756B626E — **Ascii:** UnknownBinary
Size	12 +
Parent Objects	

Data Format

```
Int32      objectType
Uns32      objectSize
EndianEnum byteOrder
RawData    objectData[objectSize]
```

SubObjects

Description

The unknown binary object is a way of transporting unknown data found in a binary file. It is an encapsulated replica of the original data found in a binary metafile, containing the object type (an Int32), the object size (in bytes), the byte order of the original file, and the data itself. The byte order is needed if unknown data is transported across different processors, and allows for parsing endian-specific primitives within the raw data block.

Unknown binary objects may be written in either the text or binary files.

When an unknown binary object is encountered in a metafile, it is up to the reading program to either:
• transport the data around
• validate it and convert it to a known object
• discard the data

Unknown objects are inherently “dirty”, meaning you may assume the unknown binary object may contain out-of-sync (bogus) information, as the original object may have been removed from its original context.

Example

```
UnknownBinary (
  1701605476
  4
  BigEndian
  0x0AB2
)
```

Unknown Text

Creation Date 4/6/95
Mod Date 4/7/95

Type	**Parent Hierarchy** Shared, Shape — Drawable Referencable
	Binary: uktx 0x756B7478 — **Ascii:** UnknownText
Size	sizeof(name) + sizeof(data)
Parent Objects	any

Data Format

```
String  asciiName
String  contents
```

SubObjects

Description

The unknown text object is a way of transporting unknown data found in a text file. It is an encapsulated replica of the original data found in a text metafile, containing the object type (a String), and a text string containing the original data. In some cases, white space and comments may have been stripped from the contents field.

Unknown text objects may be written in either the text or binary files.

When an unknown text object is encountered in a metafile, it is up to the reading program to either:
• transport the data around
• validate it and convert it to a known object
• discard the data

Unknown objects are inherently “dirty”, meaning you may assume the unknown text object may contain out-of-sync (bogus) information, as the original object may have been removed from its original context.

Example

```
UnknownText (
  “Ellipsoid”
  “”
)
```

Macintosh Path

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** Shared, Storage	Referencable
	Binary: macp 0x6D616370	**Ascii:** MacintoshPath
Size	sizeof(String)	
Parent Objects	ALWAYS: Reference	

Data Format

```
String  pathName
```

SubObjects

none

Description

The Macintosh path specifies the pathname of an external file reference using the pathname specification found in the Inside Macintosh volumes. (essentially, a colon-based separator)

Example

```
Container (
  Reference ( 43 )
  MacintoshPath ( “:::Foo:Bar:Models:Cheryl” )
)
```

Unix Path

Creation Date 10/21/94
Mod Date 3/8/95

Type	**Parent Hierarchy** Shared, Storage	Referencable
	Binary: unix 0x756E6978	**Ascii:** UnixPath
Size	sizeof(String)	
Parent Objects	ALWAYS: Reference	

Data Format

```
String  unixPath
```

SubObjects

none

Description

The unix path object serves as a way to reference files on a unix file system.

The path should obey naming standards for unix operating systems.

Example

```
Container (
  Reference ( 23 )
  UnixPath ( "./shaders.eb" )
)
```

C String

Creation Date 10/21/94
Mod Date 3/15/95

Type	**Parent Hierarchy** Shared, String	Referencable
	Binary: strc 0x73747263	**Ascii:** CString
Size	sizeof(String)	
Parent Objects		

Data Format

```
String   cString
```

SubObjects

none

Description

The CString is a way of embedding text in a metafile.

Other string types allow for internationalization.

The only allowable characters in a CString are 7-bit ASCII numbers.

The following characters may be “escaped” with the ‘\’:

Example

```
CString (
  “Copyright (c) 1994 Apple Computer, Inc.”
)
```

UNICODE

Creation Date 10/21/94
Mod Date 3/15/95

TYPE

Parent Hierarchy Shared, String

Referencable

Binary: uncd 0x756E6364

Ascii: Unicode

SIZE

4 + length * 2

PARENT OBJECTS

DATA FORMAT

```
Uns32     length
RawData   unicode[length * 2]
```

DESCRIPTION

The unicode object is another way of embedding text in a metafile.

See UNICODE reference for details.

EXAMPLE

```
Unicode (
  6
  0x457363686572
)
```

SUBOBJECTS

none

Pixmap Texture

Creation Date 10/21/94
Mod Date 1/24/95

Type

Parent Hierarchy	Shared, Texture — Referencable
Binary: txpm 0x7478706D	**Ascii:** PixmapTexture

Size

28 + rowBytes * height + padding

Parent Objects

SOMETIMES: TextureShader

Data Format

```
Uns32         width
Uns32         height
Uns32         rowBytes
Uns32         pixelSize
PixelTypeEnum pixelType
EndianEnum    bitOrder
EndianEnum    byteOrder
RawData       image[rowBytes * height]
```

• 0 < width
• 0 < height
• 0 < pixelSize < 32
• width * pixelSize ≤ rowBytes
• PixelTypeEnum is:

Binary	Text
0x00000000	RGB8
0x00000001	RGB16
0x00000002	RGB24
0x00000003	RGB32

• EndianEnum is:

Binary	Text
0x00000000	BigEndian
0x00000001	LittleEndian

SubObjects

none

Description

A generic means of transferring pixmap data. Used in the Texture Shader.

Example

```
PixmapTexture (
   256 256 # width/height
   128 # rowBytes
   32 # pixelSize
   RGB24
   BigEndian BigEndian
   0x00123232...
   0x...
)
```

● View Hints

Creation Date 1/20/95
Mod Date 2/24/95

◁ ▷

Type

Parent Hierarchy `Shared`	`Referencable`
Binary: `vwhn` `0x7677686E`	**Ascii:** `ViewHints`

Size

0

Parent Objects

none

No Data

SubObjects

```
1 Renderer (optional)
1 Camera (optional)
many Lights (optional)
1 AttributeSet (optional)
1 ImageDimensions (optional)
1 ImageMask (optional)
1 ImageClearColor (optional)
```

Description

The subobjects of the **view hints** object specifies the preferences supplied by a writing application when rendering a scene.

The semantic to be followed when a **view hints** object is encountered in the metafile is that the view hints is specified previous to a list of objects to be rendered to that particular view hints preference. The subobjects of the view hints object are inherited from the previous view hints in a metafile.

For example, if a modelling application contains 10 camera locations for viewing various portions of a scene, it would first store the default view as the first object in a metafile, then the group representing the scene, then a view containing the second camera position, then a reference to the scene, etc.

Example

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  Container (
    ViewAngleAspect ( 0.73 1.0 )
    CameraPlacement (
      0 0 30
      0 0 0
      0 1 0
    )
  )
  DirectionalLight ( -0.7 -0.7 -0.65 )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
    SpecularControl ( 3 )
  )
  ImageDimensions ( 200 200 )
)
ref1:
BeginGroup ( DisplayGroup ( ) )
...
EndGroup ( )
Container (
  ViewHints ( )
  Container (
    ViewAngleAspect ( 0.73 1.0 )
    CameraPlacement (
      0 10 0
      0 0 0
      0 1 0
    )
  )
)
Reference ( 1 )
```