# PEX Protocol Specification

*Version 5.2 - Draft Consortium Standard, rev 3*

*January 10, 1996*

Copyright © 1988, 1989, 1990, 1991, 1992, 1993, 1994 Massachusetts Institute of Technology
All rights reserved.

Permission to use, copy, modify, and distribute this document for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice are retained, and that the name of M.I.T. not be used in advertising or publicity pertaining to this document without specific, written prior permission. M.I.T. makes no representations about the suitability of this document or the protocol defined in this document for any purpose. It is provided "as is" without express or implied warranty.

Copyright © 1995, 1996 X Consortium

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.

# 1 Introduction

## 1.1 The PEX Extension to X11

PEX is an extension to the *X Window System Protocol, Version 11*[†1], which is referred to as X11 or simply X throughout this document. This specification adheres to the philosophy, requirements, and conventions of X11.

The purpose of PEX is to render as a 2D image in a X drawable a set of 3D geometrical objects.

This specification, Version 5.2, extends PEX to include most of what is commonplace in 3D graphics functionality (e.g., texture mapping, alpha blending, antialiasing, transparency, and the use of general mechanisms like accumulation buffers).

The Double-Buffering Extensions to X (DBE) is required if double-buffering of PEX rendering is supported. Requests whose size is too large to be expressed in the 16-bit request header length field are supported if the Big Request Extension to X (BRE) is supported. The X Imaging Extension (XIE) can be utilized to create and process image data, not only for the generation of images to be used as texture map bases, but also by providing filtering tools suitable for the generation of the individual levels of a color mipmap, or for blending multiple images.

## 1.2 Typographical Conventions Used in This Specification

The following typographical conventions are applied to PEX entities:

**Table 1-1 . Typographical Conventions**

| **Component** | **Delimiter** | **Font** | **Separation** | **Example** |
|---|---|---|---|---|
| Request (embedded in text) | preceding “PEX” | bold | capital letter | **PEXGetExtensionInfo** |
| Output command | | | space, capital letter | Line Color |
| Data type | | capitalized | underscore | COLOR_SPECIFIER |
| Enumerated type class | | italics | capital letter | *LineType* |
| Enumerated type value mnemonic | | italics | capital letter | *DashDot* |
| Constant value | | italics | capital letter | *AllEnabled* |
| Lookup table name | | | space | color lookup table |
| Error | | italics | capital letter | *OutputCommand* |
| Event | | italics | capital letter | *MaxHitsReached* |
| Parameter of a request or OC | | italics | underscore | *color_type* |
| Pipeline state / PC field | | italics | space | *line color* |
| Renderer attribute | | italics | space | *clip list* |
| Lookup table field | | italics | space | *approximation mode* |
| Bit name | | italics | underscore | *perspective_proj* |

†1) “X Window System” is a trademark of the Massachusetts Institute of Technology

**Table 1-1 . Typographical Conventions**

| Component | Delimiter | Font | Separation | Example |
|---|---|---|---|---|
| Coordinate values | parentheses | | comma | (1,0,0) |
| Vector | angle brackets | | comma | <1,0,0> |

See §3.6 for a description of the request specification syntax, §6.2.4 for a description of the output command specification syntax, and §7.2.1 for a description of the lookup table specification syntax.

## 1.3 History and Acknowledgments

### 1.3.1 Version 5.2

We gratefully acknowledge all those who have contributed to the completion of PEX 5.2.

Jeff Stevenson served as the Chief Architect for PEX; his leadership was a key ingredient in completing this effort. Tom Gaskins provided the first draft specification for PEX 5.2. Jane Sczechowski provided key portions of the early drafts and managed the discussion necessary for her to complete the Consortium and Public review drafts. Karl Schultz is the current PEXlib Chief Architect and has kept us honest through-out the development of this specification. In addition to these; Bruce Campbell, Iliese Chelstowski, Allan Frankel, John Fujii, Ethan Gannett, Ken Garnett, Jan Hardenbergh, Jay Hersh, Susan Hunts, Louis Khouw, France LeVangia, Pete Nishimoto, Richard Pimentel, Lou Schaffer, Robert Scheifler, Greg Stiehl, Jeff Thomas, and Steve Wampler made significant contributions to PEX 5.2. We acknowledge that others, not specifically named, also contributed.

Those mentioned above represent the many companies who supported the effort. Thanks to Hewlett-Packard, IBM, SunSoft, Digital, Evans & Sutherland, Fujitsu, Hughes, Kubota, Metheus, Network Computing Devices, Megatek, SHOGraphics and Silicon Graphics.

### 1.3.2 Versions 5.0 and 5.1

PEX began when, at the first X3D meeting of the MIT X Consortium in June 1987, DEC and Sun jointly proposed that a protocol be defined based on an initial working draft developed by DEC. The authors of this first document are recognized: Randi J. Rost, Jeffrey Friedberg, Jeffrey S. Saltz, Pete Nishimoto, and William H. Clifford, all of DEC.

The PEX architecture team met between June 1987 and August 1991. Members were: Jeffrey Friedberg, John McConnell, Pete Nishimoto, Randi J. Rost, and Jeffrey S. Saltz of DEC; Dave Gorgen, Tom Gross, and Jeff Stevenson of HP; David Plunkett of Solbourne; Jan Hardenbergh of Stardent; and Marty Hess and Jim Van Loo of Sun. Many others attended architecture team meetings and actively participated in the effort.

Acknowledgment is also extended to Bertram Herzog of the University of Michigan, chairman of X3D, who was responsible for the fair and timely process which brought PEX to fruition.

Director Robert Scheifler and Jim Fulton of the X Consortium also contributed technically and ensured PEX conforms to the spirit and definition of the X Window System.

To Randi Rost, the first PEX Document Editor, go the thanks of all. Sally Barry and Jeff Friedberg are acknowledged for their thorough and painstaking review. Sally, Document Editor of 5.0, acknowledges the invaluable assistance of Jeff Friedberg, Jan Hardenbergh, and Randi Rost. Paula Womack of DEC, Document Editor of 5.1, acknowledges Ken Garnett of Shographics.

To Todd Neman of DEC, Document Editor of the previously separate encoding specification, goes our sincere thanks. Randi Rost is thanked for serving as Document Editor from January 1989 until April 1990.

Thanks to the PEX Sample Implementation team at Sun: Marty Hess, Lisa Chabot, Cheryl Huntington, Erwin Hom, Nash Aragam, Tom Gaskins, John Recker, and Chris Nicholas, for providing useful review and sanity checks of the PEX specification.

Also acknowledged are the contributions of all the other participants not explicitly named.

# 2 Functional Overview

This chapter provides a broad overview of PEX. It defines, at a high level, the functionality implemented by the PEX extension to X. This chapter is not intended to be a complete functional specification. Instead it serves as a complementary introduction to the specification of each individual resource, request, and output command documented in subsequent chapters.

Some of PEX's roots lie in PHIGS[†1] as extended by PHIGS PLUS[†2]. Occasionally this chapter makes use of identical PHIGS or PHIGS PLUS specification by referencing specific clauses in those standards.

## 2.1 PEX Components

This section describes the major components through which the functionality in PEX is provided. The major components are output commands, drawables and their ancillary buffers, and the PEX resources: renderers, pipeline contexts, lookup tables, name sets, structures, fonts, and search contexts. Some of the functionality of PEX is discussed in the context of these components.

### 2.1.1 Output Commands

The output commands (OCs) are used to define a model -- a set of 3D objects. Each OC represents a single element of the model. Each element defines an intrinsic part of the model, such as a geometrical object, a description of some aspect of the geometry's appearance, a relationship between components of the model, an identifier, or application-specific data. There are many output commands. Some are discussed below. Many others are discussed elsewhere in this overview. When passed to a PEX renderer, a set of output commands is transformed to a 2D image in an X drawable.

PEX receives output command values independent of a particular drawable. During rendering, if a drawable cannot support the requested value, a default value is applied instead.

#### 2.1.1.1 Primitives

A primitive is an output command which defines a geometrical object. At a minimum, a geometrical object is defined by its positional components, which define the space it occupies. Additionally the definition may include aspects inherent to the positional components, such as color or edge visibility.

Some primitives include a list of floating-point data with each positional component. The interpretation of the floating-point values is driven by functionality independent of the object's definition. This allows the definition of the object to be extended as needed.

PEX has a set of defined primitive types which are specified alphabetically in Chapter 6. These primitives can be classified into the four broad areas described below.

†1) International Standard ISO/IEC 9592-1, Programmer's Hierarchical Interactive Graphics System (PHIGS).

†2) International Standard ISO/IEC 9492-4, Programmer's Hierarchical Interactive Graphics System Plus Lumière und Surfaces (PHIGS PLUS).

#### 2.1.1.1.1 Surface Primitives

Surface primitives are formed of one or more facets. A facet is formed by a set of regions in a single plane. Each region is defined by the segments joining its ordered vertices. The vertices may be explicitly defined or may be parametrically defined. It is the responsibility of the client to ensure facet vertices are coplanar. Each facet defines a two-sided surface which does not have any depth. Therefore sets of facets can be connected to define an enclosing surface such as a sphere, but they cannot be used to form a solid entity.

The surface primitives are listed in Table 6-7 (§6.5).

**Interior and Edges**

The interior of a facet is defined by the even-odd rule (§PHIGS 4.5.1) and includes the region underneath the edges of the facet. There are separate surface interior and surface edge attributes. The rendering of edges is enabled by the pipeline state’s *surface edges* field, modified with the Surface Edge Flag OC (§6.9.81). Some surface primitives optionally include a visibility flag for each edge of the primitive. When edge rendering is enabled, these individually specified edge visibility flags are used if they are available. Otherwise all of a primitive’s edges are rendered when edge rendering is enabled.

**Geometric Normal**

Each facet is either front-facing or back-facing. The facet’s geometric normal, a unit length vector, indicates the orientation of the facet. If the geometric normal, when transformed into NPC space (§2.2.1.3), has a non-negative *Z*-coordinate, the facet is front-facing.

How a facet’s geometric normal is determined is a function of the type of surface primitive. For the PEX surface primitives which are identical to PHIGS primitives, this determination is defined in PHIGS PLUS 4.5.5.2. Notice if the geometric normal is computed, the order in which the facet’s vertices are defined plays a role. The pipeline state’s *vertex order* field, modified with the Vertex Order OC (§6.9.99), allows the ordering, for the purpose of geometric normal computation, to be reversed.

The geometric normals for the arc area types is the normalized cross product of the major and minor axes (major X minor). For the elliptical arc areas, the major and minor axes are as explicitly defined by the primitive. For circular arc areas, the major axis is the MC space *X*-axis and the minor axis is the *Y*-axis.

The geometric normal of each facet of a sphere is the unit vector perpendicular to the facet and in the direction of a vector from the sphere’s center through the facet. The unit vector perpendicular to the facet and in the direction of a vector from the cone’s axis through the facet defines the geometric normal of each facet of a cone. The geometric normal of a cone whose axis endpoints are coincident is implementation-dependent.

The geometric normal for each triangle in a triangle fan is calculated identically to the geometric normal for the odd triangle strip triangles, where $P_i$ is the first vertex of the triangle fan, $P_{i+1}$ is the next to newest vertex of the triangle fan, and $P_{i+2}$ is the newest vertex of the triangle fan.

The geometric normal for each triangle in a triangle strip or triangle fan within a triangle strip fan set is calculated as for a triangle strip or triangle fan respectively. The geometric normal for an independent triangle within a triangle strip fan is calculated identically to the geometric normal for the odd triangle strip triangles, where $P_i$, $P_{i+1}$, and $P_{i+2}$ are the oldest, middle, and newest vertices defining the triangle.

**Facet Distinguishing and Culling (§PHIGS PLUS 4.5.6)**

In addition to the surface interior attributes, there are backface surface interior attributes. The backface attributes are applied to back-facing facets only if enabled by the pipeline state’s *facet distinguish flag* field, modified with the Facet Distinguish Flag OC (§6.9.31); otherwise the surface interior attributes are applied to both front- and back-facing facets. Therefore facet distinguishing indicates whether back-facing facets are distinguishable by being rendered with different surface interior attributes than the front-facing facets.

Facet culling allows facets to be thrown out, that is not rendered, based on whether the facets are front-facing or back-facing. Whether front-facing, back-facing, or no facets are culled is controlled by the pipeline state's *facet culling mode* field, modified with the Facet Culling Mode OC (§6.9.30). Facet culling is independent of facet distinguishing.

#### 2.1.1.1.2 Line Primitives

A line is defined by the sequence of connected segments joining its ordered vertices. Curve primitives conceptually define a series of vertices along the locus.

The line primitives are listed in Table 6-9 (§6.5).

#### 2.1.1.1.3 Text Primitives

A text primitive defines a planar character string.

The text primitives are listed in Table 6-2 (§6.5).

#### 2.1.1.1.4 Marker Primitives

A marker is a symbol at a given position.

The marker primitives are listed in Table 6-10 (§6.5).

### 2.1.1.2 Primitive Attributes

A primitive attribute is an output command which in some way affects the appearance of primitives. The primitive attributes are specified alphabetically in §6.9.

Some primitive attributes apply to any type of primitive, while others apply to only a subset of primitives. The definition of each type of primitive includes a list of the attribute sets (lists of primitive attributes and renderer attributes) which apply to it. Table 6-1 through Table 6-15 (§6.5) describe each attribute set and the primitives to which the attribute set applies.

### 2.1.1.3 Model Identification and Location Aids

The Pick Identifier OC (§6.10.10) sets the value of the pick identifier state which applies to all subsequent primitives. It is returned as part of the path when a primitive is picked. Pick Identifier OCs are also useful for navigation when structure editing, since the element pointer can be moved to a Pick Identifier OC with request **PEXSetElementPointerAtPickID** (§12.2.20).

The Label OC (§6.10.8) marks a location in a structure to assist in structure editing.

These two OCs do not affect PEX rendering.

### 2.1.1.4 Application Convenience

The Application Data OC (§6.10.1) allows applications to store private data. The No-op OC (§6.10.9) allows applications to create place-holder elements. These two OCs do not affect PEX rendering.

### 2.1.1.5 Sequential Processing

The Sequential Processing OC (§6.10.11) indicates whether some parallel processing may be applied to subsequent output commands or strict sequential processing is required.

### 2.1.2 Renderer

The renderer is the central PEX component. It transforms output commands representing a 3D model to a 2D image drawn in a drawable. Most of the other PEX components are bound to a renderer to accomplish this transformation.

A renderer is a PEX resource. It has many attributes which affect its rendering. The **PEXCreateRenderer** (§10.3.13), **PEXChangeRenderer** (§10.3.5), **PEXGetRendererAttributes** (§10.3.18), and **PEXFreeRenderer** (§10.3.17) requests are used to manage a renderer resource.

Request **PEXBeginRendering** (§10.3.2) associates a drawable with a renderer and initializes the renderer. Request **PEXEndRendering** (§10.3.14) completes the rendering and disassociates the renderer and the drawable. Requests **PEXInitRenderer** (§10.3.21), **PEXClearRenderer** (§10.3.6), and **PEXFlushRenderer** (§10.3.16) accomplish subsets of these operations without incurring the overhead of modifying the association between the renderer and the drawable.

Output commands are defined and sent directly to the renderer with request **PEXRenderOutputCommands** (§10.3.27) for immediate mode rendering. This request must be invoked between a pair of **PEXBeginRendering/PEXEndRendering** requests. Request **PEXRenderNetwork** (§10.3.26) causes the output commands stored in a structure network to be sent to the renderer. This is referred to as structure mode rendering. Request **PEXRenderNetwork** implicitly performs a **PEXBeginRendering/PEXEndRendering** pair.

It is possible to render structure networks even when immediate mode rendering. This is accomplished by using the execute and include structure output commands (§2.1.6.1) which, like all output commands, have the same behavior independent of whether the renderer receives them when processing output commands from **PEXRenderOutputCommands** or when traversing a structure.

With request **PEXAccumulateState** (§10.3.1), a structure network's output commands are processed up to a given location such that the renderer's pipeline state is affected, but no primitives are actually rendered. Then request **PEXRenderElements** (§10.3.25) can be used to render a range of output commands within a structure. This provides a quick way to rerender a portion of an image.

#### 2.1.2.1 Attributes

Each renderer has attributes which affect its rendering. Like the non-primitive output commands, the renderer attributes typically affect how a model is rendered, but unlike the output commands, the renderer attributes are not considered an intrinsic part of the model definition.

Renderer attributes are dynamic, that is, they take effect immediately.

The complete list of renderer attributes is described in §10.2; some of them are reviewed immediately below and throughout the rest of this overview.

##### 2.1.2.1.1 Renderer State

One of the renderer's attributes is its state (§10.2.36). When the renderer is not associated with a drawable, its state is *Idle*. The renderer's state is *Rendering* when output primitives are being rendered. There are two remaining modes, *MultipassRendering* and *Picking*.

**Multipass Rendering**

The renderer's state is *MultipassRendering* when the same set of output commands is being sent to the renderer multiple times. Transparency methods *TwoPass* and *Multipass* require multipass rendering.

If the renderer's state is not *Idle*, the *MultipassRendering* state is initiated with request **PEXInitMultipass** (§10.3.20). All output commands subsequently passed to the renderer are considered part of a pass, many of which may be required to produce the requested effects. The first pass consists of the output commands sent to the renderer between the **PEXInitMultipass** request and a **PEXNextPass** (§10.3.22) or **PEXNextPassWithoutReply** (§10.3.23) request. These latter two requests demarcate the end of the current pass and the beginning of the next. The server expects the same output commands to be sent in each pass.

One pass after another is performed until no more are needed. The number of passes required is determined by the server and conveyed to the client through the reply of **PEXNextPass**. That request returns a value indicating the number of additional passes needed, if the server can determine that, or simply that more passes are needed, or that no more are needed. Clients may use **PEXNextPass** and **PEXNextPassWithoutReply** options to abort a multipass session or to inform the server to produce the best image it can with only one more pass. When the final pass is completed, the renderer's state reverts to its value when **PEXInitMultipass** was invoked.

**Picking**

The *Picking* renderer state indicates output commands are being sent to the renderer so the renderer can identify, with the given pick method, and return to the client, one or more primitives with at least one detectable point that lies in the pick aperture. Primitives are not rendered when the state is *Picking*.

A point on a primitive is detectable (§ 11.1.1) if at the time the primitive is processed the pipeline state's *names* pass the detectability filter (§ 2.1.5.1) defined by the renderer's *pick inclusion set* and *pick exclusion set* attributes; if the pipeline state's *names* fail the invisibility filter (§ 2.1.5.3); the primitive is not culled (as controlled by the pipeline state's *facet culling mode*); and the point on the primitive is not clipped away by the model clipping volume, view clipping planes, the NPC limits (§ 2.2.1.3), the renderer's *clip list*, or the drawable's boundaries. The geometric definition of the primitive is used, so, for example, the interior of surface primitives with interior style *Hollow* or *Empty* are detectable. It is implementation-dependent whether rendering effects, such as line width, which increase the area of the primitive are considered during picking.

The pick aperture (§ 11.1.2) is specified when a pick is initiated. It is defined as a 2D location (all depths are included) and radius in DC space, or a rectangular parallelepiped defined by its opposite corners in NPC space.

A picked primitive is returned to the client in the form of a pick path. The pick path shows each level of hierarchy from the first structure to the picked primitive. Each member of the pick path consists of a structure resource identifier, offset, and pick identifier. The offset is the element position of the element invoking the next structure in the hierarchy or the element position of the picked primitive. The pick path is formed from the renderer's *current path* (§ 2.1.2.1.2) attribute and the pipeline state's *pick identifier.*

Structure mode clients wishing to pick just one primitive use request **PEXPickOne** (§ 11.2.3). Immediate mode clients use request **PEXBeginPickOne** (§ 11.2.4), followed by the **PEXRenderOutputCommands** requests, and finish with request **PEXEndPickOne** (§ 11.2.2)**.**

Structure mode clients wishing to pick all eligible primitives initialize the renderer's *pick start path* attribute indicating the location in a structure network at which to start the pick. Request **PEXPickAll** (§ 11.2.6) is then invoked. If **PEXPickAll** cannot return all the pick paths, it returns as many as it can and indicates more pick paths are available. The request also sets the *pick start path* renderer attribute to reflect the point at which it should continue the pick. The client continues invoking **PEXPickAll**, fetching pick paths until **PEXPickAll** indicates there are no more paths.

The mechanism differs slightly for immediate mode clients which wish to pick all eligible primitives. These clients invoke **PEXBeginPickAll** (§ 11.2.4). PEX begins searching for hits as soon as **PEXRenderOutputCommands** is invoked. The client ends the pick by invoking request **PEXEndPickAll** (§ 11.2.5). If **PEXEndPickAll** cannot return all the pick paths, it returns as many as it can and indicates more paths are available. The client then uses the renderer's *pick start path* attribute to determine which output commands it should send to allow PEX to resume the pick when **PEXBeginPickAll** is next invoked. Event *MaxHitsReached* is generated when the server has determined as many pick paths as it can. The client can take advantage of this event to avoid sending output commands which will have to be resent in subsequent **PEXBeginPickAll/PEXEndPickAll** rounds.

### 2.1.2.1.2 Current Path

The renderer's *current path* attribute consists of structure resource identifier and offset pairs. Initially the path has a single member with both structure identifier and offset set to zero. Each time a structure is executed or included, or a

**PEXBeginStructure** or **PEXRenderNetwork** request is received, a new pair is added to the current path, with the given structure identifier and an initial offset of zero. Whenever processing of an executed or included structure is completed, or a **PEXEndStructure** request is received, the last pair is removed from the current path. As each output command is processed, the offset of the last member of the current path is incremented.

### 2.1.2.1.3 Echo Mode

The *echo mode* renderer attribute controls whether primitives, as they are rendered, should be unechoed, echoed, or drawn normally. An echoed primitive's visual attributes are modified to emphasize the primitive. When echoing is enabled, the *echo method* (§10.2.13) and *echo color* (§10.2.12) renderer attributes indicate how the echoing is to be accomplished. If a primitive is designated to be both highlighted and echoed with color methods, it is echoed and not highlighted.

### 2.1.2.1.4 Transparency

The transparency of primitives is controlled by the *transparency method* renderer attribute. In addition to *None*, there are four transparency methods: *ScreenDoor*, *AlphaBlend*, *TwoPass*, and *Multipass*. The alpha component of each primitive's color is used in the transparency computations. An alpha value of 0.0 is transparent (i.e., zero coverage), and an alpha value of 1.0 is opaque (i.e., 100% coverage). If the primitive's color does not include an alpha component, a default alpha value is used. For surface primitives, the default alpha value is (1.0 - *transmission_coefficient*). The backface transmission coefficient is used when rendering a back-facing facet when the pipeline state's *facet distinguish flag* is *True.* The transmission coefficients are selected from the surface reflection attributes (and backface surface reflection attributes) (§2.2.2.2). For all other primitives, the default alpha value is 1.0.

No control is provided over the *ScreenDoor* transparency method, beyond selection of the method and the alpha component of each primitive's color.

The pipeline state's *alpha blend function*, modified with the Alpha Blend Function OC (§6.9.2), allows control over the alpha blending function used when the *AlphaBlend* transparency method is used. Some of the alpha blend functions require an alpha buffer (§2.1.3.2), an ancillary buffer to the drawable.

The *TwoPass* transparency method requires two passes. All primitive pixels with an alpha component of 1.0 are rendered in the first pass. All remaining primitive pixels are rendered in the second pass.

The *Multipass* transparency method is additionally controlled with the *transparency threshold* renderer attribute. The threshold is a list of *n* decreasing alpha values. A total of *n*+1 passes are needed where all primitive pixels with alpha values between the first threshold alpha value and 1.0, inclusive, are rendered in the first pass; all primitive pixels with alpha values from the *i*th threshold alpha value to the *i-1*th threshold alpha value are rendered in the *i*th pass; and all remaining primitive pixels, whose alpha values are from 0.0 to the *n*th threshold alpha value, are rendered in the final pass. The client indicates the server should determine the number of passes by setting the transparency threshold to be an empty list.

### 2.1.2.2 Pipeline State

The pipeline state is a data structure which maintains a large number of attributes which affect rendering. When a primitive is rendered, the applicable pipeline state attributes are applied to it. The pipeline state is not a renderer attribute, and its contents are not renderer attributes. Instead the pipeline state is implicitly part of the renderer. It exists only when the renderer's state is not *Idle*. The non-primitive output commands modify the pipeline state attributes. The specification of each non-primitive output command indicates which pipeline state field it modifies. Therefore these output commands do not actually modify the primitives; they modify the pipeline state which is then applied to each primitive as it is rendered.

The renderer actually maintains a stack of pipeline states. Each time an Execute Structure OC (§6.10.6), a Conditional Execute Structure OC (§6.10.3), or a **PEXBeginStructure** request (§10.3.3) initiates the traversal of a new structure, a copy of the pipeline state's attributes are saved. Its *global transform* and *global transform hints* fields are immediately set to the current composite modeling transformation (§2.2.1.1), and the *local transform* and *local transform hints* fields are immediately set to the identity matrix. Traversal of the new structure potentially causes any of the pipeline state fields to be further modified. When the execution of the new structure is terminated, because its last OC was processed, or a Conditional Return OC (§6.10.5) or **PEXEndStructure** request (§10.3.15) forced its completion, the most recently saved copy of the pipeline state is restored. This mechanism is used to create hierarchical models.

#### 2.1.2.2.1 Pipeline Context

The pipeline context is a PEX resource whose attributes match the fields of the pipeline state (§9.1.1). The *pipeline context* renderer attribute is the resource identifier of the pipeline context associated with the renderer. It is used to initialize the renderer's pipeline state whenever requests **PEXBeginRendering**, **PEXRenderNetwork**, or **PEXInitRenderer** are invoked.

The **PEXCreatePipelineContext** (§9.2.3), **PEXChangePipelineContext** (§9.2.1), **PEXCopyPipelineContext** (§9.2.2), **PEXGetPipelineContext** (§9.2.5), and **PEXFreePipelineContext** (§9.2.4) requests are used to manage a pipeline context resource.

A client can save, optionally modify, and restore the pipeline state with requests **PEXCopyPipelineStateToPC** (§10.3.9) and **PEXCopyPCToPipelineState** (§10.3.8). Once a copy of the pipeline state has been moved to a pipeline context, it can be manipulated with the pipeline context management requests.

### 2.1.3 Drawables and their Ancillary Buffers

The renderer draws 3D geometrical objects into the X drawable bound to it. The type (e.g., window, pixmap, or DBE buffer) and visual identifier (if applicable) of those drawables supported by PEX for a given screen, depth, and visual are returned by request **PEXMatchRendererTargets** (§4.1.5).

Support for the Double-Buffering Extension to X (DBE) is required for double-buffering. The Multi-Buffering Extension to X (MBX) is also specified for backwards compatibility, but support is not required. Double-buffered rendering with DBE is accomplished via the window ID and controlled via DBE or PEXlib utilities. Double-buffered rendering with MBX is accomplished by alternately binding the image buffers to a renderer.

Request **PEXGetAncillaryBufferInfo** (§4.1.1) returns the ancillary buffers, and their characteristics, supported for given visuals and/or pixmap depths. PEX uses the accumulation, alpha, and Z-buffer ancillary buffers as described below.

#### 2.1.3.1 HLHSR and the Z-buffer

Hidden line / hidden surface removal (HLHSR) algorithms are used to avoid rendering those portions of primitives that are obscured by other primitives. The Z-buffer is used for the most common HLHSR algorithms. It is used to store the depth of rendered pixels. As each new pixel is considered for rendering, its depth is compared against the Z-buffer value to determine whether the pixel is in front of previous rendering to that location and therefore should be rendered.

HLHSR is controlled by the *HLHSR mode* renderer attribute (§10.2.20). HLHSR is disabled with mode *Off*. Mode *ZBuffer* enables HLHSR using the Z-buffer. Mode *ZBufferId* also selects the Z-buffer, but indicates its usage is further controlled by the pipeline state's *HLHSR identifier*, modified with the HLHSR Identifier OC (§6.9.38). This allows HLHSR to be specified as a part of the graphical model. The HLHSR Identifier OC specifies three usages of the Z-buffer: *Disable*, *Enable*, and *EnableTestOnly*. Usage *EnableTestOnly* specifies a pixel is, as usual, compared against the Z-buffer to determine whether the pixel is rendered, but if it is determined the pixel should be rendered, the Z-buffer is not updated with the pixel's Z-value. This capability is typically used to temporarily add objects to the image, which respect depth sorting, but which are later removed by rerendering the affected area of the image.

The pipeline state's *Z-buffer test*, modified with the Z-buffer Test OC (§6.9.101), provides still further control over the usage of the Z-buffer. The Z-buffer test allows the client to choose which results of the comparison between the pixel's depth value and the Z-buffer's value result in a determination the pixel should be rendered. It is not specified in exactly which coordinate space HLHSR is accomplished, but it is guaranteed it is done in a right-handed coordinate system where a larger Z-value is closer to the eyepoint. Therefore typically the selected Z-buffer test is *GreaterThan* or *GreaterThanOrEqual*. However the Z-buffer Test OC allows any relational test to be selected. Additionally it supports *Pass*, which by causing all pixels to be rendered, in affect disables depth sorting but ensures the Z-buffer is still updated if HLHSR is enabled. The Z-buffer test is in effect whenever any HLHSR method using the Z-buffer is enabled.

Whenever an implicit or explicit **PEXBeginRendering** occurs and either the *HLHSR mode* renderer attribute selects an HLHSR method requiring a Z-buffer, or the *clear Z* renderer attribute (§ 10.2.5) is *True*, a Z-buffer is allocated, if necessary, and bound to the drawable. (A new Z-buffer is allocated only if a suitable one is not already bound to the drawable. Sometimes a drawable will have an unsuitable Z-buffer, e.g., if a resize occurred and the Z-buffer has not yet been resized.) If these conditions do not occur but the *HLHSR mode* renderer attribute is later changed with request **PEXChangeRenderer** to indicate a Z-buffer is required, the Z-buffer is allocated, if necessary, at that time. Once allocated, the Z-buffer remains bound to the drawable until the drawable is freed. As mentioned above, the buffer is reallocated as necessary.

If PEX was unable to allocate a Z-buffer, an *Alloc* error is generated, rendering is aborted, and the *renderer state* renderer attribute is set to *Idle*.

The *clear Z value* renderer attribute (§ 10.2.5) specifies the initial Z-buffer value. All Z-buffer values within the renderer's clip list (set by the *clip list* renderer attribute (§ 10.2.6)) are set to this value, if the *clear Z* renderer attribute is *True*, when a Z-buffer is allocated or whenever an implicit or explicit **PEXBeginRendering** occurs and a Z-buffer is bound to the drawable. The Z-buffer can also be cleared by invoking request **PEXClearRenderer** (§ 10.3.6) with the *clear_Z_buffer* parameter set.

The client can save and restore a drawable's Z-buffer on the server with requests **PEXCopyZBufferToPixmap** (§ 10.3.12) and **PEXCopyPixmapToZBuffer** (§ 10.3.11), which use the implementation-dependent format and range of the Z-buffer values. Request **PEXGetAncillaryBufferInfo** (§ 4.1.1) returns the type and range of values for a particular Z-buffer. The client can inquire and modify any arbitrary rectangle of the Z-buffer with requests **PEXGetZBuffer** (§ 10.3.19) and **PEXPutZBuffer** (§ 10.3.24), which represent the Z-buffer values as floating point values in the range 0.0 to 1.0.

#### 2.1.3.2 The Alpha Buffer

The alpha buffer is used by some of the alpha blending functions. It is used to store the alpha component of rendered pixels. Those alpha blending functions using the "destination alpha" value in determining the color of a pixel are using the value stored in the alpha buffer.

Whenever an implicit or explicit **PEXBeginRendering** occurs and either the *transparency method* renderer attribute selects a transparency method requiring an alpha buffer, or the *clear alpha* renderer attribute (§ 10.2.3) is *True*, an alpha buffer is allocated, if necessary, and bound to the drawable. (A new alpha buffer is allocated only if a suitable one is not already bound to the drawable. Sometimes a drawable will have an unsuitable alpha buffer, e.g., if a resize occurred and the alpha buffer has not yet been resized.) If these conditions do not occur but the *transparency method* renderer attribute is later changed with request **PEXChangeRenderer** to indicate an alpha buffer is required, the alpha buffer is allocated, if necessary, at that time. Note transparency method *AlphaBlend* does not cause an alpha buffer to be allocated unless the pipeline state's *alpha blend function* also selects a function requiring an alpha buffer. Therefore a change in the pipeline state's *alpha blend function* can also trigger the allocation of the alpha buffer. Once allocated, the alpha buffer remains bound to the drawable until the drawable is freed. As mentioned above, the buffer is reallocated as necessary.

If PEX was unable to allocate an alpha buffer, an *Alloc* error is generated, rendering is aborted, and the *renderer state* renderer attribute is set to *Idle*.

The alpha component of the *background color* renderer attribute (§ 10.2.2) specifies the initial alpha buffer value. If the background color does not have an alpha component, a default value of 1.0 (opaque) is used. All alpha buffer values within the renderer's clip list (set by the *clip list* renderer attribute (§ 10.2.6)) are set to this value, if the *clear alpha* renderer attribute is *True*, when an alpha buffer is allocated or whenever an implicit or explicit **PEXBeginRendering** occurs and an alpha buffer is bound to the drawable. The alpha buffer can also be cleared by invoking request **PEXClearRenderer** (§10.3.6) with the *clear_alpha_buffer* parameter set.

The client can save and restore a drawable's alpha buffer on the server with requests **PEXCopyAlphaBufferToPixmap** (§10.3.7) and **PEXCopyPixmapToAlphaBuffer** (§10.3.10). Request **PEXGetAncillaryBufferInfo** (§4.1.1) returns the type and range of values for a particular alpha buffer.

#### 2.1.3.3 The Accumulation Buffer

The accumulation buffer is typically used by rendering algorithms which require the entire image, or some portion of it, to be rendered multiple times and accumulated. Each accumulation buffer value corresponds to the red, green, blue, and possibly alpha components of a pixel color; therefore it is supported only by drawables whose visual class is *DirectColor* or *TrueColor*. The buffer must be at least as deep as the drawable.

The accumulation buffer is explicitly allocated and bound to a drawable with request **PEXAllocateAccumulationBuffer** (§15.3.2). It is initialized with request **PEXLoadAccumulationBuffer** (§15.3.4), which copies the contents of the drawable into its accumulation buffer. Request **PEXAccumulateBuffer** (§15.3.1) is used to subsequently add the contents of the drawable to its accumulation buffer, after multiplying the contents of both the drawable and the accumulation buffer by separate weight factors. When an algorithm has finished accumulating an image, the contents of the accumulation buffer are copied into the drawable, after multiplying the contents by a scale factor. The buffer is freed with request **PEXFreeAccumulationBuffer** (§15.3.3).

### 2.1.4 Lookup Tables

A lookup table (LUT) is a PEX resource. There are many defined types of lookup tables (§3.7.7.139). Each type supports a collection of related values. Every table potentially has multiple entries which allows a variety of combinations of the table's values to be defined.

There is a renderer attribute corresponding to a resource identifier for each table type. This is how lookup tables are bound to the renderer. During rendering, an entry of values is selected from a lookup table with an index. Typically the index is a pipeline state value, so the index is part of the graphical model's definition. The indirection of the lookup table mechanism therefore allows different sets of attribute values to be defined together and easily applied as a group to the model without modifying the definition of the model itself. The lookup table values also vary from the OC values in that the lookup table values are dependent on the capabilities of the drawable. An error is issued if an attempt is made to set a lookup table field to a value not supported by the drawable. In contrast, the setting of OC values is independent of the drawable.

The **PEXCreateLookupTable** (§7.3.3), **PEXCopyLookupTable** (§7.3.2), and **PEXFreeLookupTable** (§7.3.5) requests are used to manage a lookup table resource. Individual entries of a lookup table are set and inquired with the **PEXChangeTableValues** (§7.3.1) and **PEXGetTableValues** (§7.3.8) requests. All of an entry's values, or just a subset of the values, may be set or inquired with these requests.

Request **PEXGetTableInfo** (§7.3.7) is used to determine the total number of entries which can be defined for a type of lookup table and which entries, if any, are automatically defined upon table creation. The automatically defined entries, known as the predefined entries, are a convenience and may be modified by the client. The server may support up to 65,535 entries in a table. Even in a table which supports fewer entries, indices in the range 0 - 65,534, or 1 - 65,535, depending on the type of table, are valid. That is, the entries do not have to be consecutively indexed. Request **PEXGetDefinedIndices** (§7.3.6) returns the list of entries defined for a given table. An entry is defined either because it is a predefined entry or because request **PEXChangeTableValues** was invoked with its index. A defined entry is undefined with request **PEXDeleteTableEntries** (§7.3.4).

Since there is no guarantee a particular entry will be defined when it is accessed, a default index is defined for each type of table. If the entry selected by the default index is not defined, each type of table has default values which are used.

#### 2.1.4.1 Bundle Tables

Some of the lookup tables, referred to as bundle tables, provide an alternate means of setting attributes which are also settable through an OC. Each of these tables define a set of related line, surface, surface edge, data mapping, marker, or text attributes. However values within a bundle table are individually selected. Which source is used during rendering is controlled with the attribute's aspect source flag (ASF). If the ASF is *Individual*, the pipeline state is the source of the attribute's value. If the ASF is *Bundled*, an entry of the attribute's corresponding bundle table, selected by the bundle table's index in the pipeline state, is the source of the attribute's value. The aspect source flags themselves are part of the pipeline state, modified by the Set Individual ASF OC (§6.9.39).

The interior bundle table, which contains surface attributes, is unique in that there are two modes of accessing it. By default, the pipeline state's *interior bundle index* selects an entry which is used for all of the bundled surface attributes. However the *PHIGS interior bundle mode* renderer attribute (§10.2.30) may be used to select a different access method which matches the PHIGS PLUS surface attribute bundling. When the PHIGS interior bundle mode is *True*, five different pipeline state indices, the *interior bundle index*, *backface interior bundle index*, *reflectance bundle index*, *backface reflectance bundle index*, and *parametric surface bundle index*, are used to separately select subsets of five different entries.

### 2.1.5 Name Sets

With the pipeline state's *names* field, primitives can be labeled for the purposes of highlighting, enabling and disabling their rendering, and picking. Each name is simply an integer. Names are added and removed from the graphics model with the Add Names to Name Set (§6.9.1) and Remove Names from Name Set (§6.9.71) OCs.

A name set is a PEX resource which defines a list of names. A name set resource is managed with requests **PEXCreateNameSet** (§8.2.3), **PEXChangeNameSet** (§8.2.1), **PEXCopyNameSet** (§8.2.2), **PEXGetNameSet** (§8.2.5), and **PEXFreeNameSet** (§8.2.4). Name set resources are referenced by the pipeline context resource attributes, where they are used for initializing, setting, and inquiring the pipeline state's *names*. Name set resource identifiers are also included in the renderer attributes to define the inclusion and exclusion sets for the highlight, invisibility, and detectability filters.

#### 2.1.5.1 Filtering

When filtering is applied to a primitive, the pipeline state's *names* are compared against the filter's inclusion and exclusion sets. The primitive passes the filter if at least one of the pipeline state *names* is in the inclusion set and none of the pipeline state *names* are in the exclusion set.

#### 2.1.5.2 Highlighting

A primitive is highlighted when the pipeline state's *names* pass the highlighting filter defined by the renderer's *highlight inclusion set* and *highlight exclusion set* attributes. When a primitive is highlighted, its visual attributes are modified to emphasize it. Control over the attribute modification is provided through the highlight lookup table (§7.2.8). Each entry of this table sets a highlight method. The entry of the highlight table used to highlight a primitive is selected by the pipeline state's *highlight index*, modified with the Highlight Index OC (§6.9.37). The only way to unhighlight a primitive is to clear and rerender.

Both highlighting and echoing visually emphasize primitives. However highlighting differs from echoing in that individual primitives are selected for highlighting through filtering. Filtering is dependent on the names which are part of the graphical model's definition. Echoing, which is controlled through the *echo mode* renderer attribute, applies to all primitives rendered when the mode is *Echo*. Which primitives are echoed is independent of the graphical model's definition.

#### 2.1.5.3 Invisibility

If a primitive's pipeline state *names* pass the invisibility filter defined by the renderer's *invisibility inclusion set* and *invisibility exclusion set* attributes, the primitive is not rendered. Such a primitive is not a part of the image and is therefore not eligible for operations such as picking. By contrast, a primitive which is invisible due to being 100% transparent is still a part of the image. For example, a fully transparent primitive still affects the Z-buffer and is eligible for picking.

#### 2.1.5.4 Detectability

One of the requirements which makes a primitive eligible for picking is its detectability. To be detectable, a primitive's pipeline state's *names* must pass the detectability filter defined by the renderer's *pick inclusion set* and *pick exclusion set* attributes.

### 2.1.6 Structures

A structure is a PEX resource which stores OCs for later execution. Each output command corresponds to a single element of the structure. A structure resource is managed with requests **PEXCreateStructure** (§12.2.5), **PEXCopyStructure** (§12.2.4), and **PEXDestroyStructures** (§12.2.9).

OCs are added to a structure with request **PEXStoreElements** (§12.2.22). Elements are copied from one structure to another with request **PEXCopyElements** (§12.2.3). Elements are inquired with requests **PEXGetElementInfo** (§12.2.14) and **PEXFetchElements** (§12.2.11). Elements are deleted with requests **PEXDeleteElements** (§12.2.6), **PEXDeleteElementsBetweenLabels** (§12.2.7), and **PEXDeleteElementsToLabel** (§12.2.8).

#### 2.1.6.1 Structure Hierarchy

Structures may be related to each other hierarchically to form structure networks which are organized as acyclic graphs. The hierarchical relationships are formed with the structure control output commands, which like all output commands, are also structure elements.

The Execute Structure (§6.10.6) and Include Structure (§6.10.7) OCs invoke another structure. That is, when one of these output commands is processed, processing continues with the output commands stored in the invoked structure. Execute Structure causes the pipeline state to be saved and a new copy to be made and modified (§2.1.2.2). The new copy is deleted and the saved copy restored when processing of the invoked structure is complete. In the case of Include Structure, there is no save and restore of the pipeline state. That is, modifications to the pipeline state by the processing of the invoked structure carry through when processing of the invoking structure is later continued. The execution and inclusion of structures affect the renderer's *current path* as described in Section §2.1.2.1.2.

There are also a set of output commands which allow structures to be conditionally invoked and a conditional early return from an invoked structure. The Condition Flags (§6.10.2), Test Extent (§6.10.12), and Test Extent 2D (§6.10.13) OCs are used to modify the *condition flags* field of the pipeline state. The extent is a bounding box which is transformed to device coordinates. A clip test and a size test are conducted against the extent, and the results are used to modify the *condition flags*. The clip test is typically used to conditionally skip the processing of primitives which will be clipped. The size test is a comparison between the extent's DC space diagonal length and an extent size selected from an extent size lookup table (§7.2.7). The size test can be used to conditionally render different OCs representing different versions of the model based on the screen size of the model. The Conditional Execute Structure (§6.10.3), Conditional Include Structure (§6.10.4), and Conditional Return (§6.10.5) OCs use the condition flags to affect the flow of output commands processed.

A hierarchical model can be drawn with immediate mode rendering even if structures aren't used. Requests **PEXBeginStructure** (§10.3.3) and **PEXEndStructure** (§10.3.15) mimic the hierarchical control offered by the Execute Structure and Conditional Execute Structure OCs. **PEXBeginStructure** saves the renderer's pipeline state and updates the modeling transformation and current path. **PEXEndStructure** restores the most recently saved pipeline state and updates the modeling transformation and current path.

All structures referencing a structure, called the parents of that structure, plus all their parents and so on up to root structures, constitute the ancestors of the given structure. **PEXGetAncestors** (§12.2.12) returns a list of paths down to a specific structure. Similarly, all structures referenced by a given structure are called its children, and the children of a structure together with all their children and so on down to structures containing no structure references OCs are called its descendants. **PEXGetDescendants** (§12.2.13) returns a list of the reference path starting from a given structure, and **PEXGetStructuresInNetwork** (§12.2.16) returns a single list of all descendants.

Request **PEXChangeStructureReferences** (§12.2.2) is provided to aid in editing the hierarchy. It is used to replace all references to a structure with references to another structure.

#### 2.1.6.2 Structure Attributes

The *element pointer* structure attribute indicates a position within the structure. Each element within the structure is identified by its position. Position 1 is the first element in the structure. Elements beyond the first are assigned sequential positions. Position 0 corresponds to the location before the first element in the structure. An element's position changes when elements between it and position 0 are added or removed. The element pointer can be positioned to any element in the structure or to position 0 with one of the following requests: **PEXSetElementPointer** (§12.2.18), **PEXSetElementPointerAtLabel** (§12.2.19), or **PEXSetElementPointerAtPickID** (§12.2.20). The element pointer is also changed when elements are added or removed. The requests that add and remove elements define their effect on the element pointer. A structure's element pointer is initially 0.

If the *editing mode* structure attribute is *StructureInsert*, a group of elements added to the structure (with request **PEXStoreElements**) is inserted between two existing elements. Otherwise the editing mode is *StructureReplace* and the group of elements replaces existing elements in the structure. A structure's editing mode defaults to *StructureInsert* when the structure is created. The mode is changed with request **PEXSetEditingMode** (§12.2.17).

The *permission* structure attribute (§12.1.1.3) indicates the extent to which a server can reorganize the structure's data while maintaining the structure network. If the permission is *StructureReadWrite*, the elements and their contents are maintained exactly as they were created. If the permission is not *StructureReadWrite*, structure operations are restricted. If the permission is *StructureWriteOnly*, the server may reorganize the data of individual elements, as long as the element type and its rendering effects are not modified. Clients cannot inquire the contents of elements in a *StructureWriteOnly* structure. If the permission is *StructureLocked*, the server may also reorganize the data represented by groups of elements, again as long as their rendering effects are not modified. The only valid element positions in a locked structure are 0 and the positions of elements which reference other structures. Therefore operations which use specific element positions, such as picking, searching, partial traversals, and structure edits, are restricted on locked structures. A structure's permission is modified with request **PEXSetStructurePermission** (§12.2.21).

A structure's attributes, along with a count of its elements, the number of four-byte units necessary to store its elements, and whether it references other structures, are inquired with request **PEXGetStructureInfo (§12.2.15)**.

### 2.1.7 Fonts

A PEX font is a resource representing a text font supported by PEX. It is analogous to the X font resource, which represents text fonts supported by the X server. It is possible for the same font to be supported by both or just one of X and PEX. Requests **PEXListFonts** (§13.2.2) and **PEXListFontsWithInfo** (§13.2.3) indicate which fonts are supported by PEX.

The PEX font files are located in the same list of directories as the X11 font files. Therefore X requests **SetFontPath** and **GetFontPath** are used to respectively set and query the current font path. Servers are required to at a minimum identify each PEX font following the X Logical Font Description (XLFD) naming conventions. Request **PEXOpenFont** (§13.2.4) is used to create a PEX font resource. The association between a PEX font and its resource identifier is deleted with request **PEXCloseFont** (§13.2.1).

Each PEX font has the following information defined for it: the first, last, and default glyphs, whether all the glyphs exist, the font's properties, and whether the font can be rendered with PEX's *StrokePrecision* text precision (§6.9.95). This information can be returned for a particular PEX font resource with request **PEXQueryFont** (§13.2.5). Request **PEXListFontsWithInfo** returns this information for each font supported by PEX.

PEX fonts to be potentially applied to a single string are defined as a font group. Each font group corresponds to an entry of the text font table (§7.2.16). The pipeline state's *text font index*, modified with the Text Font Index OC (§6.9.93), selects an entry of the text font table currently bound to the renderer. The fields in this entry are applied to each text and annotation text primitive. Each of these primitives defines a single string which is a list of mono-encodings. The rendered string is formed from its mono-encodings in the order the mono-encodings are given. There is no space between the last character of a mono-encoding and the first character of the next mono-encoding beyond the spacing applied to any character in the string. For each mono-encoding, the character width is specified (8-, 16-, or 32-bit), and a font is selected from the font group selected by the pipeline state's *text font index*.

Request **PEXQueryTextExtents** (§13.2.6) returns the text local coordinate space (§2.2.1.1) extent (bounding box) and connection point for a text string with a given set of text attributes applied.

### 2.1.8 Search Context

The search context is a PEX resource which performs a view-independent pick on a geometrical object defined by a structure network. The pick is done with a structure network only; it is completely independent of a renderer resource. The search context resource is managed with requests **PEXCreateSearchContext** (§16.2.5), **PEXChangeSearchContext** (§16.2.3), **PEXCopySearchContext** (§16.2.4), **PEXGetSearchContext** (§16.2.7), and **PEXFreeSearchContext** (§16.2.6).

The pick criteria are set with the search context's attributes (§16.2.2). The WC space pick aperture is defined by the *search position* and *search distance* search context attributes. The *model clipping flag* search context attribute indicates whether the object's model clipping is applied or ignored; model clipped primitives are not eligible for selection. The *text font table* attribute is the resource identifier of the text font table used to resolve references to a text font table. The *start path* attribute indicates the structure network to search and the element after which to begin searching. The *search ceiling* attribute indicates the structure within the network at which to stop searching.

Lastly, the search context attributes include the *normal list* and *inverted list*. Each is a list of filters. The intention is the normal list acts like a detectability filter in that it selects primitives eligible for picking, and the inverted list acts like an invisibility filter in that it indicates which primitives are invisible and therefore not eligible for picking. To be selected, a primitive must pass each of the filters in the normal list and fail each of the filters in the inverted list.

Request **PEXSearchNetwork** (§16.2.8) conducts the pick. The first primitive which meets the search context criteria, if one exists, is returned.

### 2.1.9 PHIGS Workstation and Pick Measure

The PHIGS workstation (Chapter 17) and pick measure (Chapter 18) PEX resources provide functionality that is backwards compatible with PEX 5.1. They have not been updated in PEX 5.2.

### 2.1.10 Enumerated Types and Implementation-dependent Constants

Although implementations are required to support much of the functionality specified by PEX (Chapter 19), some functionality is optional. Additionally some functionality is not explicitly specified, and the implementation is allowed some freedom in how it chooses to support that functionality. The enumerated types and implementation-dependent constants allow a way for the client to programmatically determine an implementation's capabilities.

Enumerated types provide a list of supported values for a given functionality. Each type consists of its class, which names the functionality, and a list of values. The values are represented by a numerical label and a mnemonic label, which is an ISO 8859/1 string describing the value. Each enumerated type is defined with related functionality throughout the specification. They are always given in a table labeled "Enumerated Type <*class*>". The table lists the numerical label, mnemonic, and a description of each value. Request **PEXGetEnumeratedTypeInfo** (§4.1.3) is used to query the supported values for a list of enumerated types. It can optionally return the numerical label and/or mnemonic of the supported values for each inquired enumerated class.

PEX uses implementation-dependent constants to indicate whether a specific optional functionality is supported (using constants with *True*/*False* values) and to indicate implementation-dependent non-negative integer or floating-point limits. Request **PEXGetImpDepConstants** (§4.1.4) is used to query the values of a list of implementation-dependent constants. The constants are defined with this request but are referenced by related functionality throughout the specification.

## 2.2 PEX Rendering

Rendering, the process of getting the 3D geometrical model drawn as a 2D image, is accomplished through two parallel pipelines. The transformation pipeline handles the geometric transformations, from relating the model's components to each other, describing from where and how much of the model is viewed, to orienting the final image on the drawable. The rendering pipeline determines the image's color, after applying functionality such as lighting, texturing, and depth cueing.

### 2.2.1 Transformation Pipeline

#### 2.2.1.1 Model, Text Local, and World Coordinate Spaces

Primitives are defined in their own model coordinate (MC) space. This allows primitives to be specified in whatever space is convenient. To place the primitives in a common space, they are each transformed by the composite modeling transformation into world coordinate (WC) space. The *local transform* and *local transform hints* fields of the pipeline state, which define the local modeling transformation, are modified with the Local Transform with Hints (§6.9.51) and Local Transform 2D with Hints (§6.9.53) OCs. The *global transform* and *global transform hints* fields of the pipeline state, which define the global modeling transformation, are modified with the Global Transform with Hints (§6.9.34) and Global Transform 2D with Hints (§6.9.36) OCs. These transforms are used to create the composite modeling transformation. Model coordinate space, the composite modeling transformation, and world coordinate space are defined in detail by PHIGS 4.7.2.

Each Text (§6.7.35) and Text 2D (§6.7.36) primitive defines its own text local coordinate space. This space is an orientation of model coordinate space with the same scaling as MC space. The definition of the text primitives and many of the text attributes make use of the text local coordinate space. Each Annotation Text (§6.7.1) and Annotation Text 2D (§6.7.2) primitive also defines its own text local coordinate space. In this case, the text local coordinate space also has the same scaling as MC space but is specified in NPC space.

#### 2.2.1.2 Model Clipping

Model clipping allows the client to specify objects outside a volume defined in model coordinates which are not to be rendered. The pipeline state's *model clip indicator*, which indicates whether model clipping is enabled, is modified with the Model Clipping OC (§6.9.59). The Model Clipping Volume (§6.9.60) and Model Clipping Volume 2D (§6.9.61) OCs incrementally modify the *model clip volume* field of the pipeline state. The Restore Model Clipping Volume OC (§6.9.73) restores the pipeline state's *model clip volume* to the value inherited by the structure being traversed. The model clip volume is transformed by the composite modeling transformation to world coordinates where the clip actually takes place. Model clipping is defined in detail by PHIGS 4.7.2.

#### 2.2.1.3 Viewing

The pipeline state's *view index*, modified by the View Index OC (§6.9.100), selects an entry of the view table (§7.2.18). The viewing process transforms a volume of world coordinate space into a volume in normalized projection coordinate (NPC) space. Multiple views may be applied within a single image. All the views are oriented relative to each other in all three dimensions in NPC space.

The *orientation matrix* field of a view table entry defines an orientation within world coordinate space known as view reference coordinate (VRC) space. The *mapping matrix* field defines the specific volume of VRC space to be placed within a specific volume in NPC space. It additionally indicates whether the view has a parallel or perspective projection. The *clip flags* and *clip limits* fields allow the view's volume in NPC space to optionally be clipped in the x, y, and/or z directions. The *eyepoint hint* field allows the client to explicitly indicate the view's eyepoint, which is otherwise calculated by PEX.

Viewing is defined in detail by PHIGS 4.7.4 and 4.7.5.

#### 2.2.1.4 NPC Space to Device Coordinate Space

Device coordinate (DC) space is associated with the drawable. The origin of DC space is the lower left corner of the drawable. A unit in both the $x$ and $y$ directions corresponds to a pixel. Therefore the left edge of a left-most pixel of a drawable corresponds to $x = 0$; the right edge of that pixel corresponds to $x = 1$. Likewise the bottom edge of a bottom-most pixel of a drawable corresponds to $y = 0$; the top edge of that pixel corresponds to $y = 1$. $Z$ coordinates are in the range [0.0-1.0], where $z = 0$ maps to the drawable's $z$ value furthest from the viewer, and $z = 1$ maps to the value closest to the viewer. When a drawable is resized or moved, the origin of DC space becomes the lower-left corner of the drawable at its new size and position.

The *NPC subvolume* renderer attribute (§10.2.28) defines which portion of NPC space will be mapped to the drawable. The *viewport* renderer attribute (§10.2.42) defines the portion of the drawable to which this mapping will occur. The actual viewport is adjusted so the aspect ratios for $x$, $y$, and $z$ are the same in DC space as in NPC space.That is, the actual viewport is the largest parallelepiped that can fit into the defined viewport and maintain the $x$ and $y$ aspect ratios while mapping the $z$ extent of the NPC subvolume to the entire $z$ extent of the defined viewport. This means there is uniform scaling in $x$ and $y$ from NPC to DC space, but there may be some unused rendering space at the top right side of the defined viewport.

If the viewport's boundaries extend outside the drawable, rendering to the viewport is clipped by the drawable. Additionally, rendering to the viewport is clipped in the $x$ and $y$ directions as specified by the *clip list* renderer attribute (§10.2.6). The clip list is defined by a list of non-overlapping rectangles defined in DC space.

### 2.2.2 Rendering Pipeline

#### 2.2.2.1 Color

PEX colors are defined by a type and list of components. Type *Indexed* indicates at the time of rendering the color is to be taken from the color lookup table (§7.2.2) bound to the renderer. The list of components for type *Indexed* consists of a single value, an index into the color lookup table. Each entry of the color lookup table contains the definition of one color. The other color types are all color models, such as red-green-blue (RGB). The list of registered PEX color types (§3.7.6) includes a definition and valid value range for each of the components of each color type.

As a primitive enters the rendering pipeline, its intrinsic color is determined. For the surface primitives optionally including color data Table 6-5 (§6.5), this is done through data mapping. The purpose of data mapping is to specify whether the color optionally defined for each of a primitive's vertices, optionally defined for each of the primitive's facets, or the pipeline state's *surface color* or *bf surface color* is used to determine the surface primitive's intrinsic color. The pipeline state surface and backface surface colors, also known as "aspect colors", are modified by the Surface Color (§6.9.76), Surface Color Index (§6.9.77), Backface Surface Color (§6.9.13), and Backface Surface Color Index (§6.9.14) OCs. Data mapping method *None* sets the priority to vertex color (if present), overrides facet color (if present), overrides aspect color. Method *Color* allows the client to prioritize this list of sources itself. The data mapping method applied is selected from either the pipeline state or data mapping lookup table (§7.2.4) according to the pipeline state's *data map method ASF*. If the ASF is *Individual*, the pipeline state's *data map method*, modified with the Data Mapping Method OC (§6.9.27), is used. Otherwise the ASF is *Bundled* and the pipeline state's *data map index*, modified with the Data Mapping Index OC (§6.9.26), selects an entry of the data mapping lookup table. If face distinguishing is enabled and the surface to be data mapped is back-facing, the pipeline state's *bf data map method ASF, bf data map method* (modified with the Backface Data Mapping Method OC (§6.9.9)), and *bf data map index* (modified with the Backface Data Mapping Index OC (§6.9.8)) are used similarly.

For all other primitives the intrinsic color is determined similarly to data mapping method *None*; vertex color (if present), overrides facet color (if present), overrides aspect color. For example, if polyline vertex colors are present, the *i*th vertex color determines the intrinsic color of the *i*th segment. Aspect color is defined by the pipeline state's *line color*, *marker color*, *text color*, *surface color*, or *bf surface color* as appropriate. If color other than vertex or facet color is defined with the primitive, that color also overrides aspect color.

The PEX implementation actually performs rendering in the pipeline state's *rendering color model*, modified by the Rendering Color Model OC (§6.9.72). All colors entering the rendering pipeline, such as a primitive's intrinsic color, are transformed into the rendering color model. All rendering calculations are then done in the rendering color model.

### 2.2.2.2 Lighting

Lighting is the computed effect of light sources on the color of all surface primitives Table 6-7 (§6.5). It is accomplished by applying the reflectance calculation to one or more positions on the primitive being lit. The reflectance calculation considers the eyepoint, light sources, and the primitive being lit; it is independent of other objects in the image.

The light sources are defined in the light lookup table (§7.2.10). There are four types of light sources Table 7-6 (§7.2.10). Each type has a color. Additional data is needed to define the light source depending on its type. The *Ambient* light source type illuminates primitives uniformly independent of location and orientation. The *WCS_Vector* light source type simulates light emanating from infinity along a particular direction. The *WCS_Point* light source type simulates light emanating from all directions from a particular position. The *WCS_Spot* light source type simulates light emanating from a particular position within a cone of influence centered along a particular direction. Lights are enabled by placing their lookup table indices in the pipeline state's *light source state*, modified with the Light Source State OC (§6.9.41).

Suggested formulae for the reflectance calculation are given in PHIGS PLUS Annex C. The suggested light attenuation formula is updated as follows to incorporate the quadratic attenuation factor, $C_3$, which PEX additionally supports:

$$L_a = \frac{1}{C_1 + C_2\left|O_p - L_p\right| + C_3\left|O_p - L_p\right|^2} \qquad \textbf{Equation 2.1}$$

The extent of the reflectance calculation is controlled by the reflection model. If the model is *None*, no calculation is performed. Otherwise, for each enabled light source, model *Ambient* indicates only the ambient terms of the reflectance calculation are used, model *Diffuse* indicates the ambient and diffuse terms are used, and *Specular* indicates all the terms, ambient, diffuse, and specular, are used. The reflection model applied is selected from either the pipeline state or interior bundle table (§7.2.9) according to the pipeline state's *reflection model ASF.* If the ASF is *Individual*, the pipeline state's *reflection model*, modified with the Surface Reflection Model OC (§6.9.88), is used. Otherwise the ASF is *Bundled* and the pipeline state's *interior index*, modified with the Interior Bundle Index OC (§6.9.40), selects an entry of the interior bundle table from which the *surface reflection model* field is used. If face distinguishing is enabled and the surface to be lit is back-facing, the pipeline state's *bf reflection model ASF* and *bf reflection model*, and the *backface surface reflection model* field of the interior bundle table are used similarly. (Selection of a bundled interior attribute is controlled differently if the renderer's *PHIGS interior bundle mode* attribute is set.)

The factors used in the reflectance calculation are defined by the lit point's position, reflectance normal, and intrinsic color; the light source's definition; the eyepoint (§2.2.1.3); and the reflection attributes. A facet's reflectance normal defines the orientation of a facet at the point at which a reflectance calculation is to be performed. See PHIGS PLUS 4.5.5.3 for a definition of how the reflectance normal is determined for each front-facing and back-facing facet. The back-facing reflectance normal is used for all back-facing facets, independent of the face distinguishing mode.

The reflection attributes define the ambient, diffuse, and specular coefficients, and the specular color and concentration applied to the primitive. Although the transmission coefficient is included in the reflection attributes, it is used for alpha blending rather than by the reflectance calculation. The reflection attributes are like the reflection model in that they are selected from either the pipeline state or interior bundle table according to the pipeline state's *reflection attributes ASF*. Either the pipeline state's *reflection attributes*, modified with the Surface Reflection Attributes OC (§6.9.87), or the *surface reflection attributes* field of the interior bundle table is the source of the reflection attributes. If face distinguishing is enabled and the surface to be lit is back-facing, the pipeline state's *bf reflection attributes ASF* and *bf reflection attributes*, and the *backface surface reflection attributes* field of the interior bundle table are used similarly.

### 2.2.2.3 Shading

Shading is the interpolation of geometric information, color, and reflectance calculation results across a line Table 6-12 (§6.5) or surface primitive Table 6-13 (§6.5). Interpolation is defined in detail in PHIGS PLUS 4.6.4. Surface shading additionally controls the number of reflectance calculations performed on a facet. This is described briefly in this section, then diagrammed below (Figure 2-1 - Figure 2-6 ) when texture mapping is additionally considered.

Shading of linear primitives is controlled by the polyline interpolation method. There are two methods: *None*, which disables interpolation; and *Color*, which enables interpolation of color between vertices of the vertex colors (if they are present). The pipeline state's *polyline interpolation method ASF* determines whether the source of the polyline interpolation method is the pipeline state's *polyline interpolation method*, modified by the Polyline Interpolation Method OC (§6.9.68), or the *polyline interpolation method* field of the line bundle table (§7.2.11) entry selected by the pipeline state's *line bundle index* (modified by the Line Bundle Index OC (§6.9.42)).

Shading of surface primitives is controlled by the surface interpolation method. The surface interpolation method is selected similarly to the polyline interpolation method except the pipeline state's *surface interpolation method ASF*, *surface interpolation method* (modified by the Surface Interpolation Method OC (§6.9.86)), and *interior bundle index,* and the *surface interpolation method* field of the interior bundle table entry are used. If face distinguishing is enabled and the surface to be shaded is back-facing, the pipeline state's *bf surface interpolation method ASF* and *bf surface interpolation method* (modified by the Backface Surface Interpolation Method OC (§6.9.17)), and the *backface surface interpolation method* field of the interior bundle table entry are used.

The surface interpolation methods are described below. They are listed in order of increasing accuracy at the cost of increasing computation. Note their behavior is dependent on the interior style, which is selected similarly to the surface interpolation method except the pipeline state's *interior style ASF* and *interior style* (modified by the Surface Interior Style OC (§6.9.84)), and the *surface interior style* field of the interior bundle table's entry are used. In the back-facing case, the pipeline state's *bf interior style ASF* and *bf interior style* (modified by the Backface Surface Interior Style OC (§6.9.15)), and the *backface surface interior style* interior bundle table field are used.

**Surface Interpolation Method *None***

An implementation-dependent point on each facet is selected for the reflectance calculation. If vertex color is the source of the primitive's intrinsic color, the vertex colors are interpolated to this point. The single calculated reflected color is then applied to the whole facet.

If the interior style is *Hatch*, a reflectance calculation is not performed, so the surface retains its intrinsic color. If the interior style is *Pattern*, an implementation-dependent point on each cell of the projected pattern is used to conduct the reflectance calculation on each cell.

**Surface Interpolation Method *Color***

The reflectance calculation is performed at each vertex of the primitive, using the primitive's intrinsic color and reflectance normal at each vertex. The resulting reflected colors are linearly interpolated across each facet.

If the interior style is *Hatch*, reflectance calculations are not performed, so the intrinsic color of each vertex is linearly interpolated across each facet to determine the hatch color wherever the hatch is drawn. If the interior style is *Pattern,* an implementation-dependent point on each cell of the projected pattern is used to conduct the reflectance calculation on each cell.

**Surface Interpolation Method** ***DotProduct***

Any dot products needed by the reflectance calculation are calculated with the reflectance normals at each vertex. The dot products are linearly interpolated across each facet. If vertex color is the source of the primitive's intrinsic color, the vertex colors are also linearly interpolated across each facet. The reflectance calculation is performed at each interpolated position.

If the interior style is *Hatch*, reflectance calculations are not performed, so the intrinsic color of each vertex is linearly interpolated across each facet to determine the hatch color wherever the hatch is drawn. If the interior style is *Pattern*, the dot products are still calculated for each vertex and linearly interpolated across each facet. The reflectance calculation is performed at each interpolated position with the interpolated dot products and the intrinsic color defined by the projected pattern.

**Surface Interpolation Method** ***Normal***

The reflectance normal and, if it is the intrinsic color source, color of each vertex is linearly interpolated across each facet. The reflectance calculation is performed at each interpolated position.

If the interior style is *Hatch*, reflectance calculations are not performed, so the intrinsic color of each vertex is linearly interpolated across each facet to determine the hatch color wherever the hatch is drawn. If the interior style is *Pattern*, the reflectance normals are still linearly interpolated across each facet. The reflectance calculation is performed at each interpolated position with the interpolated reflectance normals and the intrinsic color defined by the projected pattern.

### 2.2.2.4 Texture Mapping

Texture mapping simulates material properties for surface primitives Table 6-13 (§6.5) by adding visual detail with color. This is accomplished by blending the sampling of a map of colors with the primitive's color.

Texture mapping is enabled by setting the interior style to *Texture*. When face distinguishing is enabled and a primitive is back-facing, the backface interior style is used instead. If none of the other texture mapping controls have been set, by default a black and white checkerboard texture will be applied to surface primitives.

Texture maps are defined by creating texture map PEX resources. Only color mipmap texture maps are supported. A color mipmap is a pre-computed area sampling mechanism which fixes the cost of approximating the average color over a large number of texels (the individual components of a texture map). It is created as a pyramid of down-sampled levels, from the "base" to the typically single-texel "pinnacle". Each level is a 1D, 2D, or 3D array of texels, according to the dimension of the texture map. The texel types are luminance, luminance and alpha, red-green-blue, and red-green-blue-alpha, in a variety of data formats. A texture map is created with request **PEXCreateColorMipmapTM** (§14.1.1), which accepts the color mipmap levels as arrays of texels. Alternately, request **PEXCreateColorMipmapTMFromResources** (§14.1.2) creates a color mipmap from a list of resources, such as drawables, image buffers, pixmaps, or photomaps, where each resource defines one level of the mipmap. A second list of resources is used if the texels include an alpha component.

A texture map resource is enabled by referencing it from an entry of the texture mapping lookup table (§7.2.17) and enabling that entry with the pipeline state's *active textures*, modified with the Surface Active Textures OC (§6.9.74). The corresponding *bf active textures* pipeline state field, modified with the Backface Surface Active Textures OC (§6.9.12), is applied to back-facing surfaces when face distinguishing is enabled. Multiple texture maps may be selected for application.

**The Texture Mapping Lookup Table**

There are many controls which are applied to a texture map. Texture maps are referenced through a lookup table to allow multiple combinations of control values to be easily applied to the same texture map. The table mechanism also facilitates ensuring control values, many of which are dependent on the specific texture map, are valid.

The *rendering control* field (§7.2.17.1) of the texture mapping lookup table controls whether the texture is applied before or after the primitive is lit.

Several of the fields control parameterization, that is, how texels are mapped to a facet. The *parameterization method* field (§7.2.17.2) specifies how the texture map coordinates for each vertex are determined. The texture map coordinates select a position in the texture map. The portion of a texture map selected by the vertices's texture map coordinates is then applied to the facet. The *VertexCoordMC, VertexCoordWC, VertexNormalMC, VertexNormalWC, NURBUV,* and *FloatData* parameterization methods use the vertex, normal, u-v parameterization, or floating-point data list data directly as the texture map coordinates. Field *fp data index* of the texture mapping lookup table defines the index within the floating-point data list of the first of the consecutive texture mapping coordinates when the parameterization method is *FloatData.* The *ReflectionSphereVRC* and *ReflectionSphereWC* parameterization methods select texture map coordinates by picking them off a conceptual sphere surrounding the facet. A special reflection vector calculated for each vertex is used to select a point on the sphere. The resulting texture mapping is often called environment mapping since the texturing is view dependent - the facet appears to reflect fixed objects as the facet is moved around. The *reflection matrix and hints* field of the texture mapping lookup table is applied only when one of these environment mapping parameterization methods is selected. The reflection matrix is used to transform the reflection vectors before they select the texture coordinates from the sphere, which has a fixed orientation.

Independent of the parameterization method used, the texture map coordinates are lastly transformed by the orientation matrix defined by the *orientation matrix and hints* field of the texture mapping lookup table. After this transformation, a vertex's texture map coordinates are ready to select a position on the texture map.

Another group of fields in the texture mapping lookup table control the sampling of the color mipmap. When one pixel corresponds to multiple texels, the *min sample method* field (§7.2.17.4.1) indicates how the texel data is sampled down to derive a single color value. The *mag sample method* field controls the opposite case of one pixel corresponding to less than a single texel. The sample methods support a wide range of sampling options to determine the color value, from *SingleBase*, which uses a single texel in the base level of the texture map, to *WeightedBetweenMipmaps*, where the weighted averages of $2^n$ texels on each of the two levels nearest the sample depth are determined, then interpolated to the sample depth. The *sampling hints* field (§7.2.17.4.4) potentially adjusts the sampling depth, making the texturing more detailed or blurred than it otherwise would have been. It is also used to indicate blurring is unacceptably high due to low spatial frequency in a particular direction in the color mipmap.

Textures sometimes map to something less than the entire facet. The *t0, t1,* and *t2 boundary condition* fields specify how the region in each direction outside the mapped texture is handled. Condition *Wrap* indicates the texture is repeated across the facet. Condition *Mirror* also indicates the texture is repeated, but alternately in reverse order. Condition *ClampBorder* indicates the closest border texel is used to determine the sampled color value beyond that border. Condition *ClampExplicit* indicates the *boundary clamp method* field is used to determine the boundary condition. (This level of indirection exists to prevent ambiguity in regions which extend outside the mapped texture in more than one direction.) The *Discontinue* boundary clamp method indicates texturing is to be discontinued beyond the mapped texture, that is, the primitive's color before applying that texture map is preserved. Boundary clamp method *Color* indicates the *boundary clamp color* field specifies the texture's color value.

The remaining fields in the texture mapping lookup table specify the blending of the color values derived from the texture map and boundary conditions with the primitive's color. The *composition method* (§7.2.17.3) field specifies how this blending is done. Method *Replace* indicates the primitive's color is replaced with the derived texture color. The alpha component is set to 1.0 if the texture map didn't include an alpha component. Method *Modulate* specifies the primitive's color and derived texture color are multiplied. Method *BlendEnvColor* indicates the derived texture color is multiplied by the color specified in the *environment color* field. The result is added to the multiplication of the primitive's color and the inverse of the derived texture color. The primitive's alpha component is just multiplied by the derived texture alpha component (1.0 if not present). Method *Decal* specifies the derived texture color is multiplied by its alpha component. The result is added to the multiplication of the primitive's color and the inverse of the derived texture alpha component. The primitive's alpha component is not modified. Method *DecalBackground* is identical to method *Decal* except the color specified in the *background color* field is used instead of the primitive's color.

### Texture Mapping Attribute OCs

There are additional texture mapping controls which are both independent of specific textures and applicable to all the textures applied to a primitive. These are implemented as OCs.

The pipeline state’s *tm perspective correction* field, modified by the Texture Mapping Perspective Correction OC (§6.9.96), determines if, and how, texture coordinate values are adjusted when the view is a perspective projection. The choices are not to correct, to correct at each vertex, and to correct at each pixel.

The pipeline state’s *tm sample frequency* field, modified by the Texture Mapping Sampling Frequency OC (§6.9.98), specifies the frequency at which the texture map is sampled. This can be set to sample at each pixel, or to base the sample frequency on the surface interpolation method, ranging between sampling once per facet for interpolation method *None*, to once per vertex of method *Color*, to once per pixel for methods *DotProduct* and *Normal*.

The *tm resource hints* field of the pipeline state, modified by the Texture Mapping Resource Hints OC (§6.9.97), allow the client to suggest how the server could most efficiently manage the texture map resources. The client expresses a preference for whether *Speed* or *Space* is more critical, or whether it doesn’t matter. Also the client can provide a prioritized usage list of texture mapping table entries.

#### 2.2.2.5 Depth Cueing

Depth cueing modifies a primitive’s color as a function of the primitive’s *Z*-coordinates in NPC space. Typically it is used to darken, or blend with the background color, objects further away from the viewer. This heightens the visual cues which indicate the relative depth of objects in the image. Depth cueing applies to all primitives.

During rendering, an entry of the depth cueing lookup table (§7.2.5) is always selected by the pipeline state’s *depth cue index*, modified by the Depth Cue Index OC (§6.9.28). Depth cueing is actually enabled or disabled by the *depth cue mode* field of the selected depth cueing lookup table. If enabled, the *front* and *back reference plane* fields define constant *Z*-value planes in NPC space. Typically the color in the *depth cue color* field is blended with the primitive’s color in front of the front reference plane according to the multiplicative factor defined in the *front scale factor* field. The *back scale factor* field defines the factor used for the blend behind the back reference plane. These factors are interpolated between the planes to determine the blending in that region. Suggested depth cueing formulae are given in Annex E of PHIGS PLUS.

#### 2.2.2.6 Data Mapping / Lighting / Shading / Texture Mapping / Depth Cueing Diagrams

These components of the rendering pipeline are not really independent. For example, the shading method determines how often and where the reflectance calculation is performed; the shading method can determine the texture map sampling frequency; and the *rendering control* field of a texture mapping lookup table entry determines whether that texture mapping occurs before or after a primitive is lit. Figure 2-1 through Figure 2-6 illustrate this portion of the rendering pipeline.

#### 2.2.2.7 Antialiasing

Antialiasing is a blending operation used to smooth points, lines, and edges. The pipeline state’s *primitive antialiasing* field, modified with the Primitive Antialiasing OC (§6.9.69), selects which classes of primitives are to be antialiasing and which blending function is to be used. The existing base of antialiasing implementations is diverse. This is reflected in the specification of the antialiasing functionality which is heavily implementation-dependent.

#### 2.2.2.8 Color Approximation

After functionality such as lighting, shading, depth cueing, transparency, antialiasing, and HLHSR has been applied to the image’s primitives, it is necessary to convert the primitives’s color from the rendering color model to X pixel values for rendering into the drawable. This is accomplished with color approximation.

**Figure 2-1** *.Surface Interpolation Method None and Texture Mapping Sample Frequency InterpDep (per facet)*

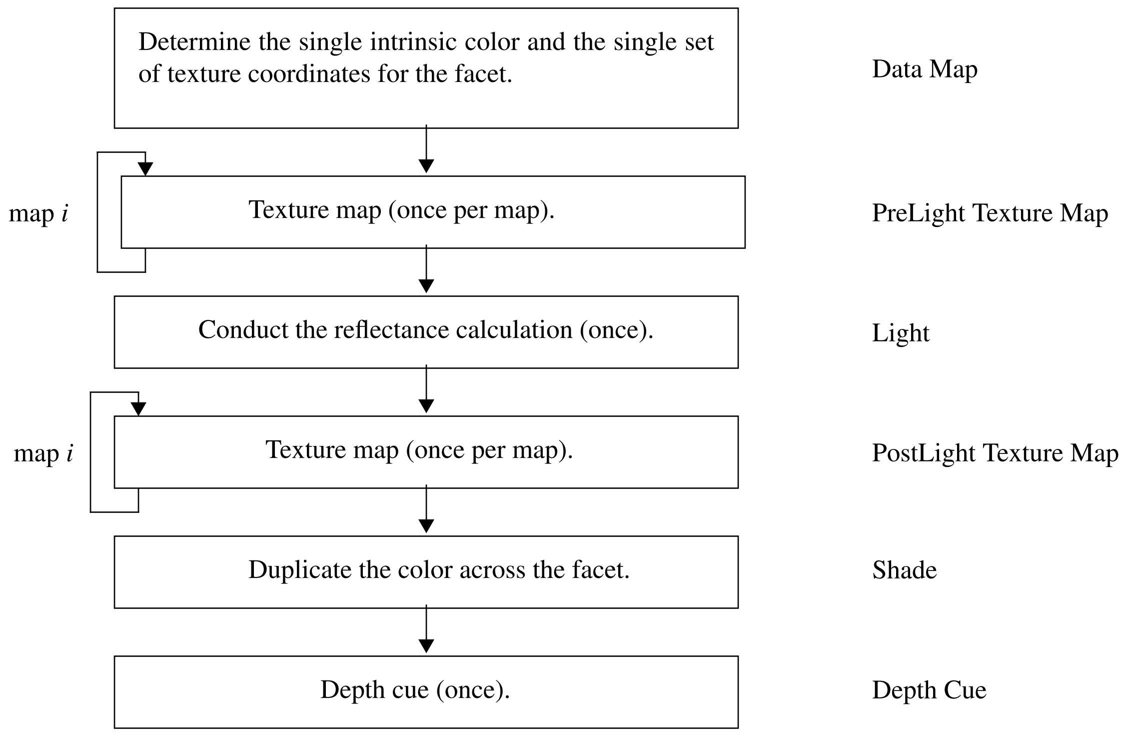


**Figure 2-2** *.Surface Interpolation Method None and Texture Mapping Sample Frequency Pixel*

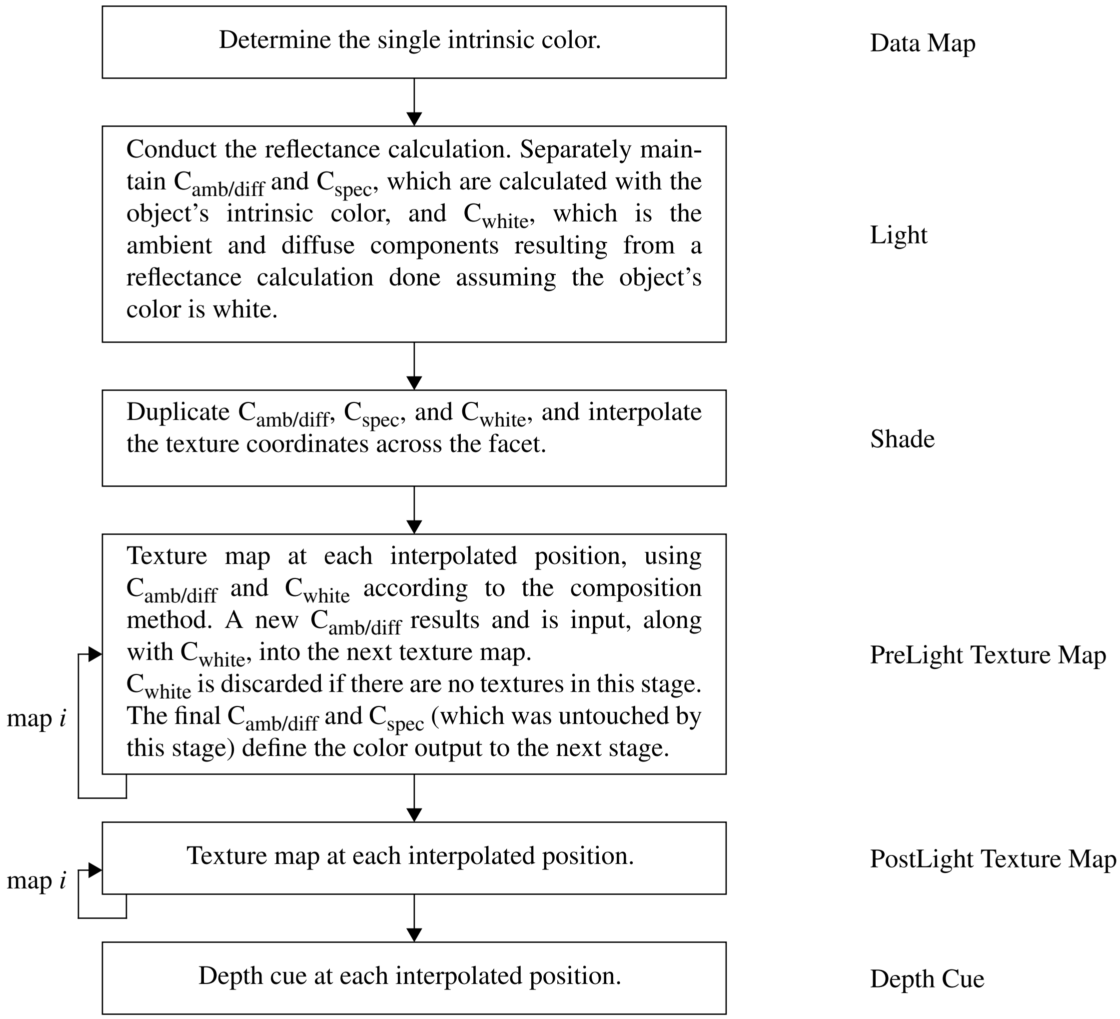


**Figure 2-3** *.Surface Interpolation Method Color and Texture Mapping Sample Frequency InterpDep (per vertex)*

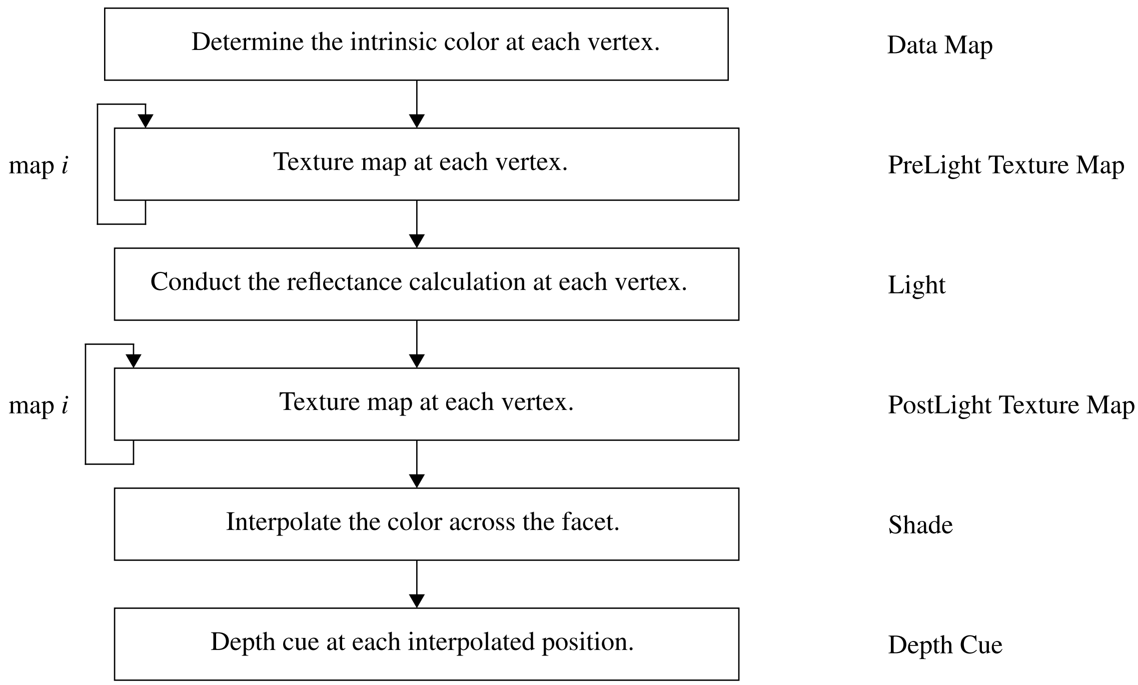


**Figure 2-4** .*Surface Interpolation Method Color and Texture Mapping Sample Frequency Pixel*

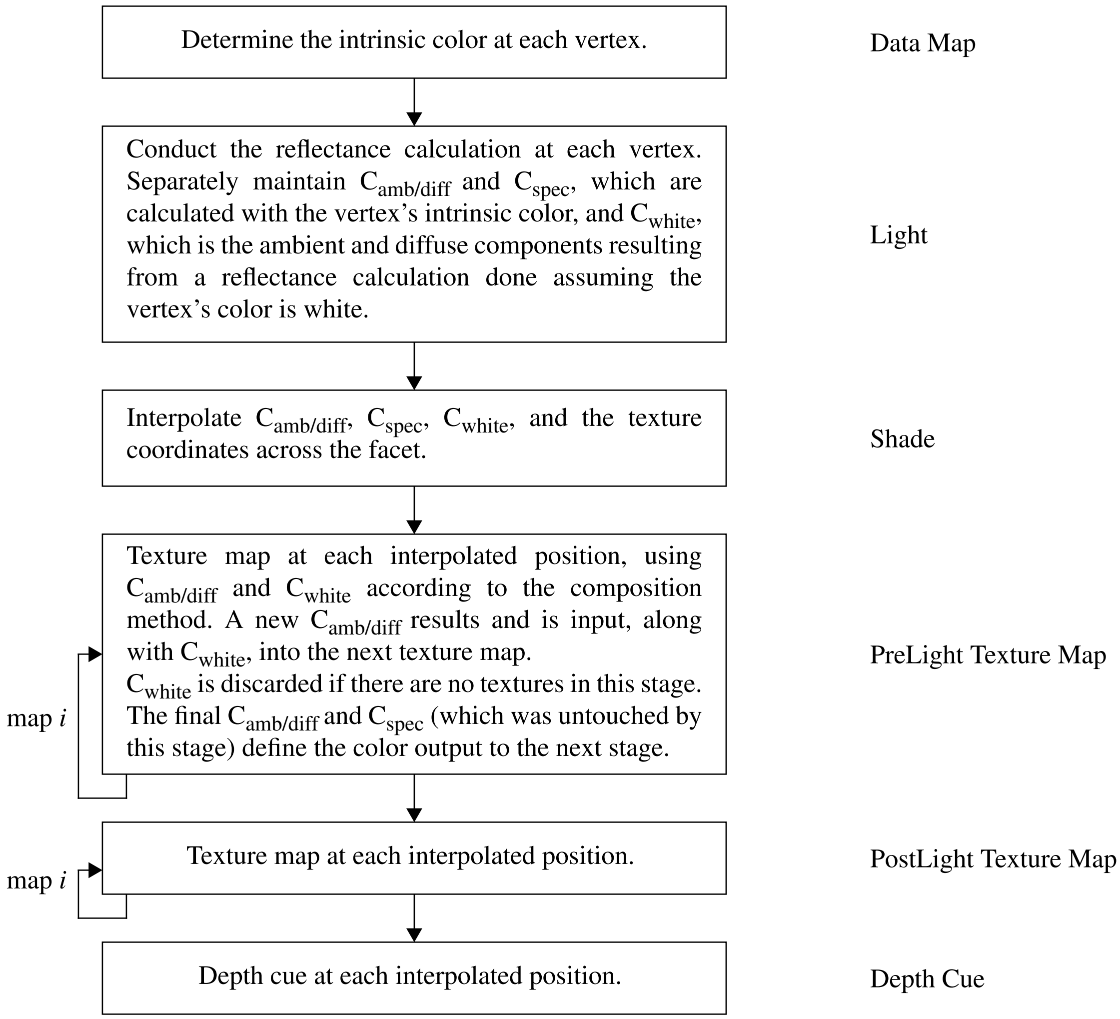


**Figure 2-5** *.Surface Interpolation Method DotProduct and Texture Mapping Sample Frequency Pixel*

**Texture Mapping Sample Frequency *InterpDep* is Also Per Pixel and is Therefore Identical**

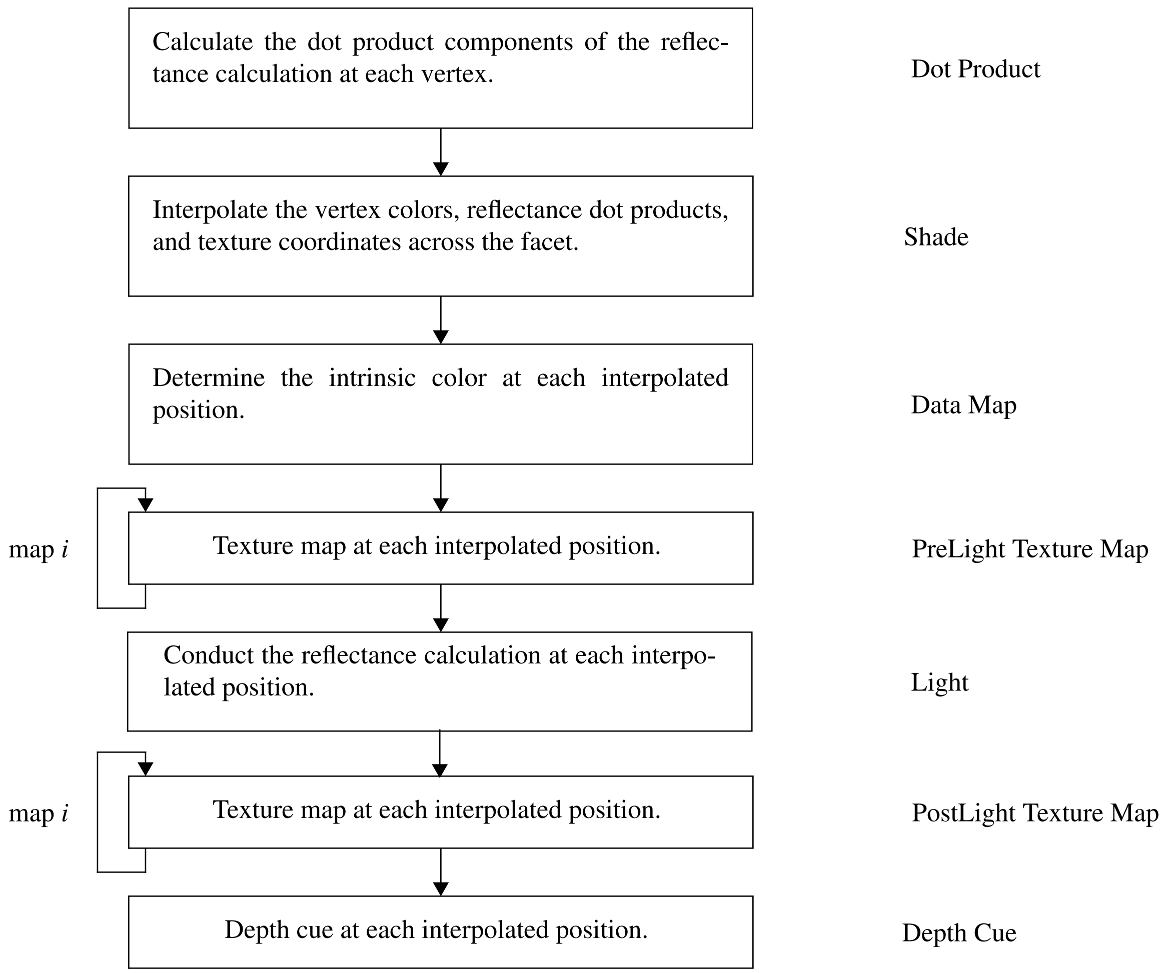
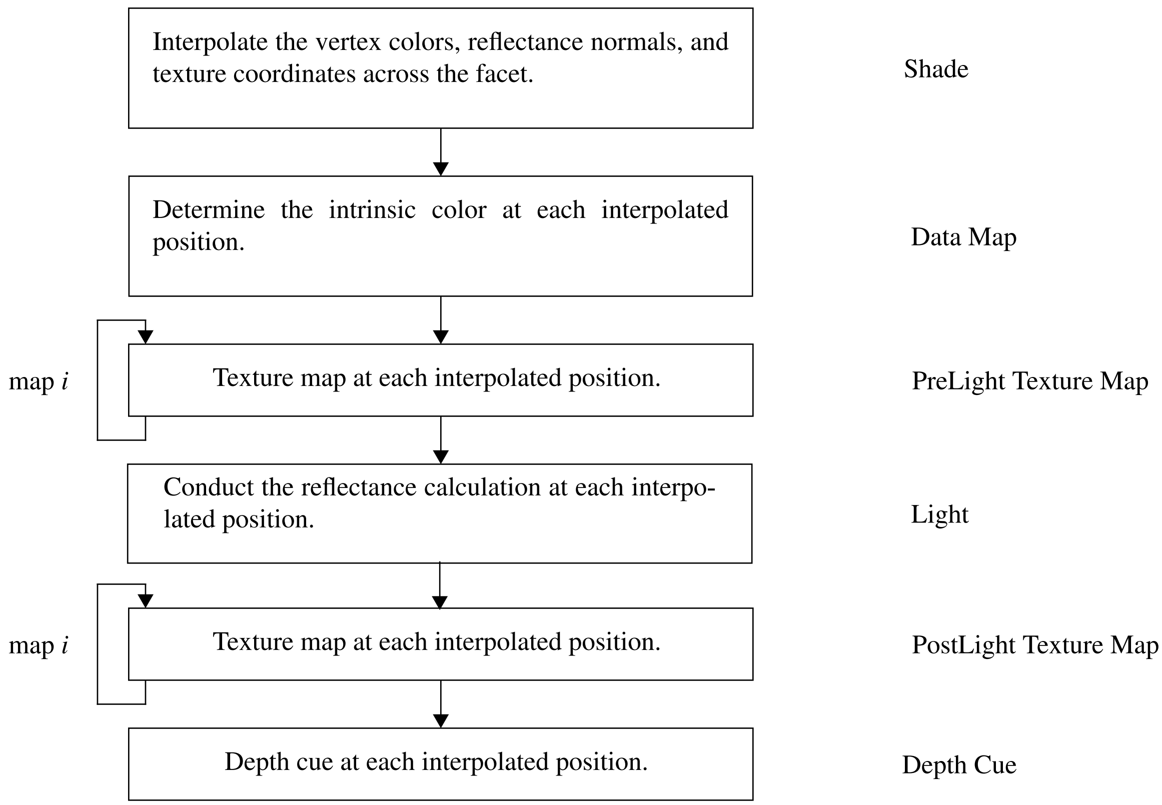


**Figure 2-6** .***Surface Interpolation Method Normal and Texture Mapping Sample Frequency Pixel***

**Texture Mapping Sample Frequency *InterpDep* is Also Per Pixel and is Therefore Identical**




An entry of the color approximation lookup table (§7.2.3) is always selected by the pipeline state's *color approx index* field, modified by the Color Approximation Index OC (§6.9.24). The primitive's rendering color model color is first transformed to the color model specified by the *color model* field of the color approximation entry. This is the color model then used to convert the color from color model type and components to pixel values. The *approximation method* field specifies how this is done.

The *ColorSpace* approximation method is intended for use with *DirectColor, TrueColor, PseudoColor,* and *StaticColor* visuals. It scales each of the color components with the *max1, max2,* and *max3* fields, which typically indicate the number of samplings (minus 1) of each color component in the section of the colormap organized as a color cube. The scaled components are then each shifted with the *mult1, mult2*, and *mult3* fields and added to the *base pixel* field to determine the pixel value.

The *ColorRange* approximation method is intended for use with *GrayScale* and *StaticGray* visuals. It scales each of the color components with the *weight1, weight2,* and *weight3* fields, which typically convert the color to a single intensity percentage. This value is multiplied by *max1*, typically the number of samplings in the colormap (minus 1), to scale the intensity value to the colormap. Lastly the intensity value is shifted with the *mult1, mult2,* and *mult3* fields, which are typically [1,0,0], and added to the *base pixel* field to determine the pixel value.

The color approximation table also includes a *dither* field. This field is a hint indicating whether the client would like dithering to be applied to enhance color resolution.

Not all PEX implementations support arbitrary color approximation settings. To address this, servers supporting the PEX extension possess either or both of the RGB_BEST_MAP and RGB_DEFAULT_MAP X standard colormap properties. These properties contain entries providing descriptions and identifiers of colormaps that are supported for PEX color approximation. PEX clients can use these properties to find colormaps to share, and/or to derive supported PEX color approximation settings. In all respects, the content and usage of these properties under PEX conform to the Inter-client Communication Conventions Manual (ICCCM).

Request **PEXQueryColorApproximation** (§4.1.6) is also provided to help a client select an appropriate color approximation that is supported. Given an example drawable and a desired color approximation, the request either indicates the color approximation is supported or returns a list of alternate supported color approximations.

#### 2.2.2.9 Pixel Color Mode

The *pixel color mode* renderer attribute (§10.2.35) is a switch enabling applications to use pixel values throughout the rendering pipeline as opposed to the color type and components used to define a primitive's intrinsic color and the rendering color model used for intermediate rendering pipeline stages. When the pixel color mode is *On*, PEX assumes the component of all type *Indexed* colors is a pixel value rather than a color lookup table index. PEX converts all colors of other types to pixel values by finding the closest match to the given color in the colormap of the drawable bound to the renderer.

Use of the pixel color mode disables much of PEX's functionality. For example, data mapping, shading, lighting, texture mapping, depth cueing, transparency, and antialiasing are disabled. The pixel color mode is useful when the client wishes to render primitives in their intrinsic pixel values without modification. This saves computation and allows the client to share the colormap with other applications. A client can easily render an image in either this simplistic mode or with full PEX functionality by defining all colors with type *Indexed* and assigning the color lookup table bound to the renderer with the same values as the colormap of the drawable bound to the renderer.

#### 2.2.2.10 Plane Masking

PEX's plane masking is identical to X's plane masking. The pipeline state's *plane mask* field, modified with the Plane Mask OC (§6.9.67), indicates which planes of the image buffer are effected by rendering.

#### 2.2.2.11 Drawing Function

PEX's drawing functions are identical to X's graphics context functions. The drawing function, which specifies how a pixel value is combined with the pixel value already in the image buffer, is specified by the pipeline state's *drawing function* field, modified with the Drawing Function OC (§6.9.29).

#### 2.2.2.12 Rendering Pipeline Caveats

Since in many implementations the following functionality uses the same resources, or creates ambiguous data, the enabling of more than one of HLHSR, transparency (including alpha blending), primitive antialiasing, the drawing function (other than *GXcopy)*, and the plane mask (other than all the planes) has implementation-dependent results. To elaborate, both multipass transparency and HLHSR may use the same buffer. It is difficult to apply HLHSR to antialiasing and alpha blending because it is unclear how a "partially covered" pixel should be reflected in the Z-buffer. Alpha blending and antialiasing may be accomplished with the same hardware and therefore interact with one another. Some implementations disable the drawing function and plane masking when alpha blending or antialiasing.

Portions of the rendering pipeline stages described in this chapter are implementation-dependent. Enumerated types and implementation-dependent constants are defined to help notify the client of the level of support. This chapter discusses the specified capabilities of PEX; the reader should review the material in the rest of this specification to understand what is guaranteed and what optional functionality can be programmatically determined.

## 2.3 Extensibility

Vendors may choose to extend their implementations of PEX beyond this specification. Each X Consortium vendor wishing to define their own extensions is given a unique vendor identifier (a number) by the MIT registry. Vendor identifiers range from 1 to 126. Identifiers 0 and 127 are reserved and will not be distributed. If all vendor identifiers are distributed, a new vendor wishing a number will have to negotiate with an existing registered vendor and share its identifier.

It is strongly recommended the following mechanisms be used:

- New requests, and extensions or modifications to existing requests, are implemented through the **PEXEscape** (§5.1.3) and **PEXEscapeWithReply** (§5.1.4) requests.
- New primitives, and extensions or modifications to existing primitives, are implemented through the GDP (§6.7.15) and GDP2D (§6.7.16) OCs.
- New OCs, and extensions or modifications to existing OCs, other than primitive OCs, are implemented through the GSE OC (§6.9.32).
- New values of an enumerated type class, and new enumerated type classes, are advertised through request **PEXGetEnumeratedTypeInfo** (§4.1.3).
- New implementation-dependent constants are advertised through request **PEXGetImpDepConstants** (§4.1.4).
- Existing events and errors are used since there is no way to programmatically advertise extensions.
- Implementations thoroughly document the interaction of extensions with each other and specified PEX functionality, including additions to the scope of PEX as defined by this chapter.

## 2.4 Compatibility Commitment

It is the client's responsibility to invoke request **PEXGetExtensionInfo** (§4.1.2) to determine the PEX version supported by the server. The client should not attempt to use functionality introduced in a more recent version than the server supports.

Clients intended for an older version of PEX than supported by the server are guaranteed to work without modification on a newer version of PEX with the following caveats: protocol which generated an error in the intended version may behave differently; and previously unspecified protocol (such as undefined bitmask and enumerated type values, or request and OC opcodes) may have a specified behavior in the newer version.

Implementation-dependent behavior is not guaranteed to be consistent even across the same vendor's releases.

# 3 Protocol Format, Data Types, and Conventions

## 3.1 Extension Identification

This extension is known as the PEX extension. Its existence in an X11 server is indicated by the string “X3D-PEX” returned by the X request **ListExtensions**. The same string is used in the X request **QueryExtension** to determine the existence of the extension.

The symbols 5.0, 5.1, and 5.2 are used throughout to indicate in which version of PEX a particular functionality was defined. If a symbol is absent, it should be assumed the functionality originated in version 5.0.

## 3.2 Request Processing

Requests and output commands (§6.1) are executed in sequential order for a given connection. Requests that process output commands are not guaranteed to be atomic with respect to concurrent rendering to the same destination drawable. However by default atomicity of execution is guaranteed for most individual output commands. See the Sequential Processing output command (§6.10.11) which allows control over the atomicity of execution over the individual output commands.

## 3.3 Protocol Format

PEX protocol formats are based on the formats specified by the *X Window System Protocol, Version 11*, referred to as X11 throughout this document. The PEX protocol adheres to the philosophy and requirements of X11.

### 3.3.1 Padding

The alignment of certain values to 2- or 4-byte boundaries may necessitate the insertion of pad bytes in some requests and replies. The encodings defined in this specification indicate where these pad bytes are needed. When the amount of padding is dependent on other values in the request or reply, such as after a variable-length list, the notation *pad(*$\beta$*)* indicates in the encoding that the required padding is that needed to pad $\beta$ to a 4-byte boundary. The values contained in pad bytes are undefined.

## 3.4 Request Format

Every X11 request contains a 4-byte header that contains an 8-bit major opcode, an 8-bit data field, and a 16-bit length field. The header is followed by zero or more additional bytes of data, where the number of additional bytes is indicated by the length field. Since PEX is a proper extension of X11, the 8-bit major opcode contains the opcode assigned to the PEX extension by the X11 server. The length field contains the length of the request, including the header, in units of four bytes. The data field contains the PEX opcode for the request.

### 3.4.1 Large Requests

If the server supports the Big Request Extension to X (BRE), requests larger than can be indicated by the 16-bit length field can be defined. BRE specifies if the request header’s length field contains 0, the request header is followed immediately by a 32-bit value giving the actual request length. The length indicates the number of 4-byte quantities in the request, including the header, BRE length field, and request data. The request data follows this length value. The major opcode and PEX opcode in a large request are contained in the request header as they are in normal-length requests.

## 3.5 Reply Format

A reply consists of at least 32 bytes, led by a 7-byte reply header consisting of a 32-bit length field, a 16-bit sequence number field whose value is assigned by the core X11 protocol, and an 8-bit type field. Zero or more additional bytes follow the header. The length field gives the length of any data following the 32 byte reply and is in units of four bytes. The length is 0 if there is no data beyond the minimum 32 bytes of the reply. The sequence number field contains the least significant 16 bits of the sequence number of the corresponding request. The type field defines the type of reply generated. The values within the unused bytes of a reply are undefined.

## 3.6 Request Specification Syntax

Curly braces {...} enclose a set of alternatives. Square brackets [...] enclose a list of components. When embedded in descriptions, request names are printed in boldface (e.g., **PEXCreateStructure**). Request parameters are lower case, use the underscore (_) for separation, and are printed in italics (e.g., *item_mask*). Defined constants, defined enumerated type mnemonics, or alternative values have an initial capital letter, may use capital letters for separation, and are printed in italics (e.g., *RGBFloat*). Defined types are printed in all caps, use the underscore for separation, and are printed in the standard font (e.g., COORD_3D).

Requests are described as follows:

**Name**

**PEXSampleRequest**

**Request**

| | |
|---|---|
| *arg1_name*: | data type |
| *arg2_name*: | data type |

**Reply**

| | |
|---|---|
| *reply_value1_name*: | data type |
| *reply_value2_name*: | data type |

No reply description is given if the request has no reply.

**Errors**

The errors potentially raised by the request, and the situations under which each error arises.

**Description**

The functional description of the request.

**Encoding:**

The protocol encoding.

### 3.6.1 Conventions Used in Forming Request Names

PEX requests use a consistent naming convention. The verbs that are commonly used in request names are described here.

| | |
|---|---|
| **Create** | Create an instance of a resource |
| **Free** | Mark a resource as no longer accessible by clients, and deallocate the system resources it uses (e.g. memory) if it is not referenced by any other resource |
| **Copy** | Copy attributes from one resource to another of the same type |
| **Get** | Return resource attributes from the server to the client |
| **Change** | Modify selected attributes of a resource |
| **Set** | Modify all attributes of a resource |
| **Destroy** | Remove a resource and all references to it, and deallocate any memory associated with it |
| **Delete** | Remove some portion of a resource |
| **Fetch** | Return structure elements from the server to the client |
| **Store** | Send structure elements from the client to a structure resource in the server |
| **Begin** | Perform an initialization of some kind |
| **End** | Perform a termination of some kind |

## 3.7 Data Types

PEX defines many data types beyond those of the core X11 protocol. The PEX data types and their encodings are described in this section. Additionally PEX data types which are used by only one or two requests or output commands are defined with those requests or output commands.

### 3.7.1 LISTofFOO

A data type name of the form LISTofFOO denotes a counted list of elements of type FOO. In cases where the number of items in the list is easily computed from information in the request or reply header, a value explicitly giving the number of items in the list is not contained in the request or reply. In all other cases (except LISTofVALUE), the number of items in the list is explicit.

### 3.7.2 LISTofLISTofFOO

A data type name of the form LISTofLISTofFOO indicates a list of lists. In the containing request or reply is a parameter giving the number of lists in the list. Each individual list within the type is preceded by a CARD32 indicating how many elements are in the list. The following sample encoding gives an example.

**Encoding: LISTofLISTofFOO**

| # of Bytes | Data Type | Description |
|---|---|---|
| 4 | CARD32 | number of lists |
| … | | potentially other things here |
| 4 | CARD32 | number of items in first list |
| *n* | FOO | first item of first list |
| *n* | FOO | second item of first list |
| … | | |

**Encoding: LISTofLISTofFOO**

| # of Bytes | Data Type | Description |
|---|---|---|
| *n* | FOO | last item of first list |
| 4 | CARD32 | number of items in second list |
| *n* | FOO | first item of second list |
| … | | |

### 3.7.3 BITMASK and LISTofVALUE in Conjunction

Several requests use the pair of data types BITMASK and LISTofVALUE in conjunction. These requests have parameters of the form:

item_mask: BITMASK[*n*]

item_list: LISTofVALUE

The *item_list* gives a list of values chosen from the set of those possible for the request. The *item_mask* specifies which of the possible values are provided in *item_list*. Each possible value is assigned a unique bit position in the bitmask. If a bit in *item_mask* is set to 1, then the corresponding value is contained in the *item_list*. Items in the list are ordered according to the position of the corresponding bit in the bitmask, from least to most significant bit. The representation of BITMASK[*n*] may contain more bits than there are defined values; unused bits in *item_mask* must be zero (or a *Value* error is generated).

The items in *item_list* may be of different data types and are padded to 4-byte boundaries. If an item occupies fewer than 4 bytes, it is contained in the least significant bits of the 4-byte word. If the item is a signed value, it is not sign extended, so the upper bytes should be masked off. The entire 4 bytes is byte-swapped as a single CARD32.

### 3.7.4 BITMASK Extension

If it becomes necessary for PEX to extend a bitmask beyond its defined length, the most significant bit is specified as a “chain” bit. If it is set, another 4 bytes of BITMASK follows on the next 4-byte boundary. Instances of chained bitmasks are explicitly noted in this specification.

### 3.7.5 Floating Point Format - FLOAT

Requests that accept or return floating-point values have within them a parameter indicating the floating-point format used within that request and in the corresponding reply. All floating-point values contained in such a request or reply are in the indicated floating-point format. Floating-point parameters are indicated by the data type FLOAT. Values of type FLOAT are defined to be in the floating-point format of their containing request or reply.

Formulae for computing the size of parameters are given in the protocol encoding descriptions. In these formulae, the variable *fp* stands for the number of bytes occupied by a single floating-point value. That number is dependent on the floating-point format indicated in the request. The size of each defined floating-point format is listed in Table 3-1 .

The floating-point formats defined by PEX are listed in Table 3-1 . The mnemonics for use with enumerated type *FloatFormat* for request **PEXGetEnumeratedTypeInfo** (§4.1.3) are also listed. The server solely determines which formats it recognizes. Those recognized can be determined via the **PEXGetEnumeratedTypeInfo** request. The server’s native floating-point format will be returned as first entry in the list of supported floating point formats returned by **PEXGetEnumeratedTypeInfo**.

**Table 3-1 . Enumerated Type *FloatFormat***

| Value | Mnemonic | Size (bytes) | Description |
|---|---|---|---|
| 1 | *IEEE_754_32* | 4 | IEEE 754 standard 32-bit floating point. 5.0 |
| 2 | *DEC_F_Floating* | 4 | DEC F (single precision) floating point. 5.0 |
| 3 | *IEEE_754_64* | 8 | IEEE 754 standard 64-bit floating point. 5.0 |
| 4 | *DEC_D_Floating* | 8 | A DEC D (double precision) floating point. 5.0 |

### 3.7.6 Colors

Colors are specified as a color type and a color value. The color type indicates whether the color is to be determined indirectly through a color lookup table or is specified directly in some color model. The defined color types are listed in Table 3-2 . The mnemonics for use with enumerated type *ColorType* for request **PEXGetEnumeratedTypeInfo** (§4.1.3) are also listed.

Servers may store direct color values in an implementation-dependent format, but they must convert those values back to the originally-specified color type when the color is subsequently queried by the client.

Except for the *Indexed* color type, if a server supports a color type, then it also implicitly supports the homogeneous version of that color type for use in color splines in NURBS curves and surfaces.

**Table 3-2 . Enumerated Type *ColorType***

| Type | Mnemonic | Color Value Data Type | Description |
|---|---|---|---|
| 0 | *Indexed* | TABLE_INDEX | An indirect color. The color value is of type TABLE_INDEX. The actual color value is taken from the color lookup table entry indicated by the table index during rendering. 5.0 |
| 1 | *RGBFloat* | COLOR_RGB_FLOAT | A direct color in the red-green-blue color model. The color value contains three floating-point values in the range [0-1] giving the respective proportions of red, green, and blue. 5.0 |
| 2 | *CIEFloat* | COLOR_CIE_FLOAT | A direct color in the CIELUV color model. The color value contains three floating-point values in the range [0-1] giving the u, v, and luminance CIE components. 5.0 |
| 3 | *HSVFloat* | COLOR_HSV_FLOAT | A direct color in the hue-saturation-value color model. The color value contains three floating-point values in the range [0-1] giving the respective proportions of hue (as a fraction of a full circle), saturation, and value. 5.0 |
| 4 | *HLSFloat* | COLOR_HLS_FLOAT | A direct color in the hue-lightness-saturation color model. The color value contains three floating-point values in the range [0-1] giving the respective proportions of hue (as a fraction of a full circle), lightness, and saturation. 5.0 |
| 5 | *RGBInt8* | COLOR_RGB_INT8 | A direct color in the red-green-blue color model. The color value contains three integer values in the range [0-255] giving the respective proportions of red, green, and blue. 5.0 |

**Table 3-2 . Enumerated Type *ColorType***

| Type | Mnemonic | Color Value Data Type | Description |
|---|---|---|---|
| 6 | *RGBInt16* | COLOR_RGB_INT16 | A direct color in the red-green-blue color model. The color value contains three integer values in the range [0-65535] giving the respective proportions of red, green, and blue. 5.0 |
| 7 | *RGBAFloat* | COLOR_RGBA_FLOAT | A direct color in the red-green-blue color model and containing an alpha value. The color value contains three floating-point values in the range [0-1] giving the respective proportions of red, green, and blue, and an alpha value in the range [0-1]†1. 5.2 |
| 8 | *RGBAInt8* | COLOR_RGBA_INT8 | A direct color in the red-green-blue color model and containing an alpha value. The color value contains three integer values in the range [0-255] giving the respective proportions of red, green, and blue, and an alpha value in the range [0-255]. 5.2 |
| 9 | *RGBAInt16* | COLOR_RGBA_INT16 | A direct color in the red-green-blue color model and containing an alpha value. The color value contains three integer values in the range [0-65535] giving the respective proportions of red, green, and blue, and an alpha value in the range [0-65535]. 5.2 |

†1) An alpha value of 0.0 is transparent (i.e., zero coverage). An alpha value of 1.0 is opaque (i.e., 100% coverage).

### 3.7.7 Definition and Encoding of Common Data Types

#### 3.7.7.1 ALPHA_BLEND_FUNC

ALPHA_BLEND_FUNC : ENUM_TYPE_INDEX (See §6.9.2 for the defined values of *AlphaBlendingFunction*.)

#### 3.7.7.2 ATEXT_STYLE

ATEXT_STYLE : ENUM_TYPE_INDEX (See §6.9.6 for the defined values of *ATextStyle*.)

#### 3.7.7.3 BITMASK

BITMASK : CARD32 (In a particular instance of BITMASK, its representation may contain more bits than there are defined values. Unused bits must be zero (or PEX generates a *Value* error).

#### 3.7.7.4 BITMASK_SHORT

BITMASK_SHORT : CARD16

#### 3.7.7.5 BOOLEAN

BOOLEAN : {*False*, *True*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | boolean |
| | 0 *False* 5.0 | |
| | 1 *True* 5.0 | |

#### 3.7.7.6 BOOLEAN16

BOOLEAN16 : {*False*, *True*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | boolean |
| | 0 *False* 5.0 | |
| | 1 *True* 5.0 | |

#### 3.7.7.7 BUFFER_ID

BUFFER_ID : RESOURCE_ID

#### 3.7.7.8 CARD8

CARD8 : unsigned 8-bit integer

#### 3.7.7.9 CARD16

CARD16 : unsigned 16-bit integer

#### 3.7.7.10 CARD32

CARD32 : unsigned 32-bit integer

### 3.7.7.11 CHARACTER

CHARACTER : {CARD8, CARD16, CARD32}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | byte character |
| Or | | |
| 2 | CARD16 | short character |
| Or | | |
| 4 | CARD32 | long character |

### 3.7.7.12 COLOR

COLOR : {INDEXED_COLOR, DIRECT_COLOR}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *sc*†1 | | color value |

†1) *sc* is 4 if the color type is *Indexed*, *RGBInt8*, or *RGBAInt8*. *sc* is 8 if the color type is *RGBInt16* or *RGBAInt16*. *sc* is 3**fp* if the color type is *RGBFloat*, *CIEFloat*, *HSVFloat*, or *HLSFloat*. *sc* is 4**fp* if the color type is *RGBAFloat.*

### 3.7.7.13 COLOR_APPROX_MODEL

COLOR_APPROX_MODEL: ENUM_TYPE_INDEX (See Table 7-2 for the defined values of *ColorApproxModel*.)

### 3.7.7.14 COLOR_APPROX_TYPE

COLOR_APPROX_TYPE : ENUM_TYPE_INDEX (See Table 7-3 for the defined values of *ColorApproxType*.)

### 3.7.7.15 COLOR_CIE_FLOAT

COLOR_CIE_FLOAT : [u, v, luminance : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | u |
| *fp* | FLOAT | v |
| *fp* | FLOAT | luminance |

### 3.7.7.16 COLOR_HLS_FLOAT

COLOR_HLS_FLOAT : [hue, lightness, saturation : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | hue |
| *fp* | FLOAT | lightness |
| *fp* | FLOAT | saturation |

### 3.7.7.17 COLOR_HSV_FLOAT

COLOR_HSV_FLOAT : [hue, saturation, value : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | hue |
| *fp* | FLOAT | saturation |
| *fp* | FLOAT | value |

### 3.7.7.18 COLOR_RGB_FLOAT

COLOR_RGB_FLOAT : [red, green, blue : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | red |
| *fp* | FLOAT | green |
| *fp* | FLOAT | blue |

### 3.7.7.19 COLOR_RGB_INT8

COLOR_RGB_INT8 : [red, green, blue : INT8]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | INT8 | red |
| 1 | INT8 | green |
| 1 | INT8 | blue |
| 1 | | unused |

### 3.7.7.20 COLOR_RGB_INT16

COLOR_RGB_INT16 : [red, green, blue : INT16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | INT16 | red |
| 2 | INT16 | green |
| 2 | INT16 | blue |
| 2 | | unused |

### 3.7.7.21 COLOR_RGBA_FLOAT

COLOR_RGBA_FLOAT : [red, green, blue, alpha : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | red |
| *fp* | FLOAT | green |
| *fp* | FLOAT | blue |
| *fp* | FLOAT | alpha |

### 3.7.7.22 COLOR_RGBA_INT8

COLOR_RGBA_INT8 : [red, green, blue, alpha : INT8]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | INT8 | red |
| 1 | INT8 | green |
| 1 | INT8 | blue |
| 1 | INT8 | alpha |

### 3.7.7.23 COLOR_RGBA_INT16

COLOR_RGBA_INT16 : [red, green, blue, alpha : INT16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | INT16 | red |
| 2 | INT16 | green |
| 2 | INT16 | blue |
| 2 | INT16 | alpha |

### 3.7.7.24 COLOR_SPECIFIER

COLOR_SPECIFIER : [color_type : COLOR_TYPE,
color_value : COLOR]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | COLOR_TYPE | color type |
| 2 | | unused |
| *sc* | COLOR | color value (see COLOR for *sc*) |

### 3.7.7.25 COLOR_TYPE

COLOR_TYPE : ENUM_TYPE_INDEX (See §3.7.6 for the defined values of *ColorType*.)

### 3.7.7.26 CONDITION_TEST

CONDITION_TEST : {*AllEnabled*, *AllDisabled*, *NotAllDisabled*, *NotAllEnabled, AlwaysPass, AlwaysFail*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | condition flags test[†1] |
| | 0 | *AllEnabled* 5.2 | |
| | 1 | *AllDisabled* 5.2 | |
| | 2 | *NotAllDisabled* 5.2 | |
| | 3 | *NotAllEnabled* 5.2 | |
| | 4 | *AlwaysPass* 5.2 | |
| | 5 | *AlwaysFail* 5.2 | |

†1) See (§6.10.3) for a detailed description of the condition flags test values.

### 3.7.7.27 CONTOUR

CONTOUR : {*Disjoint*, *Nested*, *Intersecting*, *Unknown*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | contour hint |
| | 0 | *Disjoint* 5.0 | |
| | 1 | *Nested* 5.0 | |
| | 2 | *Intersecting* 5.0 | |
| | 3 | *Unknown* 5.0 | |

### 3.7.7.28 COORD

COORD : {COORD_2D, COORD_3D, COORD_4D}

### 3.7.7.29 COORD_TYPE

COORD_TYPE : {*Rational*, *NonRational*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | spline rationality |
| | 0 *Rational* 5.0 | |
| | 1 *NonRational* 5.0 | |

### 3.7.7.30 COORD_2D

COORD_2D : [x, y : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | x |
| *fp* | FLOAT | y |

### 3.7.7.31 COORD_3D

COORD_3D : [x, y, z : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | x |
| *fp* | FLOAT | y |
| *fp* | FLOAT | z |

### 3.7.7.32 COORD_4D

COORD_4D : [wx, wy, wz, w : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | wx |
| *fp* | FLOAT | wy |
| *fp* | FLOAT | wz |
| *fp* | FLOAT | w |

#### 3.7.7.33 CULL_MODE

CULL_MODE : {*None*, *BackFaces*, *FrontFaces*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | cull mode |
| | 0 *None* 5.0 | |
| | 1 *BackFaces* 5.0 | |
| | 2 *FrontFaces* 5.0 | |

#### 3.7.7.34 CURVE_APPROX

CURVE_APPROX : [approx_method : CURVE_APPROX_METHOD,
tolerance : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CURVE_APPROX_METHOD | curve approximation method |
| 2 | | unused |
| *fp* | FLOAT | tolerance |

#### 3.7.7.35 CURVE_APPROX_METHOD

CURVE_APPROX_METHOD :
ENUM_TYPE_INDEX (See §6.9.25 for the defined values of *CurveApproxMethod*.)

#### 3.7.7.36 DATA_MAP_METHOD

DATA_MAP_METHOD : ENUM_TYPE_INDEX (See §6.9.27 for the defined values of *DataMappingMethod*.)

### 3.7.7.37 DATA_MAP_METHOD_AND_DATA

DATA_MAP_METHOD_AND_DATA :
[method : DATA_MAP_METHOD,
source_selectors : LISTofCARD16]

**Encoding:**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| 2 | DATA_MAP_METHOD | data mapping method |
| 2 | CARD16 | number of source selectors (*n*) |
| 2**n* | LISTofCARD16 | prioritized list of source selectors |
| | 0 *SurfaceColor* | |
| | 1 *VertexColor* | |
| | 2 *FacetColor* | |
| *p* | | *p*=pad(*n*) |

†1) Size *d* of a DATA_MAP_METHOD_AND_DATA is 4+2**n*+*p*.

### 3.7.7.38 DC_HIT_BOX_DATA

DC_HIT_BOX_DATA : [pick_position : DEVICE_COORD_2D,
pick_distance : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | DEVICE_COORD_2D | pick position |
| *fp* | FLOAT | pick distance |

### 3.7.7.39 DEVICE_COORD

DEVICE_COORD : [x, y : INT16,
z : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | INT16 | x |
| 2 | INT16 | y |
| *fp* | FLOAT | z |

### 3.7.7.40 DEVICE_COORD_2D

DEVICE_COORD_2D : [x, y : INT16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | INT16 | x |
| 2 | INT16 | y |

### 3.7.7.41 DEVICE_RECT

DEVICE_RECT : [xmin, ymin, xmax, ymax : INT16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | INT16 | xmin |
| 2 | INT16 | ymin |
| 2 | INT16 | xmax |
| 2 | INT16 | ymax |

### 3.7.7.42 DIRECT_COLOR

DIRECT_COLOR : {COLOR_RGB_FLOAT, COLOR_CIE_FLOAT, COLOR_HSV_FLOAT, COLOR_HLS_FLOAT, COLOR_RGB_INT8, COLOR_RGB_INT16, COLOR_RGBA_FLOAT, COLOR_RGBA_INT8, COLOR_RGBA_INT16}

### 3.7.7.43 DRAWABLE_ID

DRAWABLE_ID : {WINDOW_ID, PIXMAP_ID, BUFFER_ID}

### 3.7.7.44 DRAWING_FUNCTION

DRAWING_FUNCTION : {*GXclear*, *GXand*, *GXandReverse*, *GXcopy*, *GXandInverted*, *GXnoop*, *GXxor*, *GXor*, *GXnor*, *GXequiv*, *GXinvert*, *GXorReverse*, *GXcopyInverted*, *GXorInverted*, *GXnand*, *GXset*}[†1]

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | drawing function |
| | 0 | *GXclear* 5.2 | |
| | 1 | *GXand* 5.2 | |
| | 2 | *GXandReverse* 5.2 | |
| | 3 | *GXcopy* 5.2 | |
| | 4 | *GXandInverted* 5.2 | |
| | 5 | *GXnoop* 5.2 | |
| | 6 | *GXxor* 5.2 | |
| | 7 | *GXor* 5.2 | |
| | 8 | *GXnor* 5.2 | |
| | 9 | *GXequiv* 5.2 | |
| | 10 | *GXinvert* 5.2 | |
| | 11 | *GXorReverse* 5.2 | |
| | 12 | *GXcopyInverted* 5.2 | |
| | 13 | *GXorInverted* 5.2 | |
| | 14 | *GXnand* 5.2 | |
| | 15 | *GXset* 5.2 | |

### 3.7.7.45 ECHO_MODE

ECHO_MODE : {*NoEcho*, *Echo*, *UnEcho*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 4[†1] | CARD16 | | echo mode |
| | 0 | *NoEcho* 5.1 | |
| | 1 | *Echo* 5.1 | |
| | 2 | *UnEcho* 5.1 | |

†1) The CARD16 value is placed in the least significant portion of the four bytes.

†1) These values are those defined by the core X protocol.

### 3.7.7.46 ELEMENT_POS

ELEMENT_POS : [whence : {*Beginning*, *Current*, *End*},
offset : INT32]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | whence |
| | 0 *Beginning* 5.0 | |
| | 1 *Current* 5.0 | |
| | 2 *End* 5.0 | |
| 2 | | unused |
| 4 | INT32 | offset |

### 3.7.7.47 ELEMENT_RANGE

ELEMENT_RANGE : [position1, position2 : ELEMENT_POS]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 8 | ELEMENT_POS | position 1 |
| 8 | ELEMENT_POS | position 2 |

### 3.7.7.48 ELEMENT_REF

ELEMENT_REF : [structure_id : STRUCTURE_ID,
offset : CARD32]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | STRUCTURE_ID | structure identifier |
| 4 | CARD32 | offset |

### 3.7.7.49 ELEMENT_TYPE

ELEMENT_TYPE : CARD16

**Encoding: ELEMENT_TYPE**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | element type |
| | 0 *OCAll*†1 | |
| | 1 Marker Type | |
| | 2 Marker Scale Factor | |
| | 3 Marker Color Index | |

**Encoding: ELEMENT_TYPE**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| | 4 | Marker Color | |
| | 5 | Marker Bundle Index | |
| | 6 | Text Font Index | |
| | 7 | Text Precision | |
| | 8 | Character Expansion | |
| | 9 | Character Spacing | |
| | 10 | Text Color Index | |
| | 11 | Text Color | |
| | 12 | Character Height | |
| | 13 | Character Up Vector | |
| | 14 | Text Path | |
| | 15 | Text Alignment | |
| | 16 | Annotation Text Height | |
| | 17 | Annotation Text Up Vector | |
| | 18 | Annotation Text Path | |
| | 19 | Annotation Text Alignment | |
| | 20 | Annotation Text Style | |
| | 21 | Text Bundle Index | |
| | 22 | Line Type | |
| | 23 | Line Width | |
| | 24 | Line Color Index | |
| | 25 | Line Color | |
| | 26 | Curve Approximation Method | |
| | 27 | Polyline Interpolation Method | |
| | 28 | Line Bundle Index | |
| | 29 | Surface Interior Style | |
| | 30 | Surface Interior Style Index | |
| | 31 | Surface Color Index | |
| | 32 | Surface Color | |
| | 33 | Surface Reflection Attributes | |
| | 34 | Surface Reflection Model | |
| | 35 | Surface Interpolation Method | |
| | 36 | Backface Surface Interior Style | |
| | 37 | Backface Surface Interior Style Index | |
| | 38 | Backface Surface Color Index | |
| | 39 | Backface Surface Color | |
| | 40 | Backface Surface Reflection Attributes | |
| | 41 | Backface Surface Reflection Model | |
| | 42 | Backface Surface Interpolation Method | |
| | 43 | Surface Approximation Method | |
| | 44 | Facet Culling Mode | |
| | 45 | Facet Distinguish Flag | |
| | 46 | Pattern Size | |
| | 47 | Pattern Reference Point | |

**Encoding: ELEMENT_TYPE**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| | 48 | Pattern Reference Point and Vectors | |
| | 49 | Interior Bundle Index | |
| | 50 | Surface Edge Flag | |
| | 51 | Surface Edge Type | |
| | 52 | Surface Edge Width | |
| | 53 | Surface Edge Color Index | |
| | 54 | Surface Edge Color | |
| | 55 | Surface Edge Bundle Index | |
| | 56 | Individual ASF | |
| | 57 | Local Transform | |
| | 58 | Local Transform 2D | |
| | 59 | Global Transform | |
| | 60 | Global Transform 2D | |
| | 61 | Model Clipping | |
| | 62 | Model Clipping Volume | |
| | 63 | Model Clipping Volume 2D | |
| | 64 | Restore Model Clipping Volume | |
| | 65 | View Index | |
| | 66 | Light Source State | |
| | 67 | Depth Cue Index | |
| | 68 | Pick ID | |
| | 69 | HLHSR Identifier | |
| | 70 | Color Approximation Index | |
| | 71 | Rendering Color Model | |
| | 72 | Parametric Surface Characteristics | |
| | 73 | Add Names to Name Set | |
| | 74 | Remove Names from Name Set | |
| | 75 | Execute Structure | |
| | 76 | Label | |
| | 77 | Application Data | |
| | 78 | GSE | |
| | 79 | Marker | |
| | 80 | Marker 2D | |
| | 81 | Text | |
| | 82 | Text 2D | |
| | 83 | Annotation Text | |
| | 84 | Annotation Text 2D | |
| | 85 | Polyline | |
| | 86 | Polyline 2D | |
| | 87 | Polyline Set with Data (V5.0) | |
| | 88 | Non-Uniform B-spline Curve | |
| | 89 | Fill Area | |
| | 90 | Fill Area 2D | |
| | 91 | Fill Area with Data | |

**Encoding: ELEMENT_TYPE**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| | 92 | Fill Area Set | |
| | 93 | Fill Area Set 2D | |
| | 94 | Fill Area Set with Data (V5.0) | |
| | 95 | Triangle Strip with Data (V5.0) | |
| | 96 | Quadrilateral Mesh with Data (V5.0) | |
| | 97 | Set of Fill Area Sets with Data (V5.0) | |
| | 98 | Non-uniform B-spline Surface | |
| | 99 | Cell Array | |
| | 100 | Cell Array 2D | |
| | 101 | Extended Cell Array | |
| | 102 | GDP | |
| | 103 | GDP 2D | |
| | 104 | No-op | |
| | 105 | Arc Area Set with Data | |
| | 106 | Arc Set with Data | |
| | 107 | Cone Set with Data | |
| | 108 | Marker with Data | |
| | 109 | Non-uniform B-spline Curve with Color | |
| | 110 | Non-uniform B-spline Surface with Data | |
| | 111 | Sphere Set with Data | |
| | 112 | Triangle Fan with Data | |
| | 113 | Triangle Set with Data | |
| | 114 | Triangle Strip Fan Set with Data | |
| | 115 | Alpha Blend Function | |
| | 116 | Backface Data Mapping Index | |
| | 117 | Backface Data Mapping Method | |
| | 118 | Backface Interior Bundle Index | |
| | 119 | Backface Reflectance Bundle Index | |
| | 120 | Data Mapping Index | |
| | 121 | Data Mapping Method | |
| | 122 | Drawing Function | |
| | 123 | Sequential Processing | |
| | 124 | Highlight Index | |
| | 125 | Line Cap Style | |
| | 126 | Line Join Style | |
| | 127 | Line Miter Limit | |
| | 128 | Parametric Surface Bundle Index | |
| | 129 | Plane Mask | |
| | 130 | Primitive Antialiasing | |
| | 131 | Reflectance Bundle Index | |
| | 132 | Vertex Order | |
| | 133 | Z-buffer Test | |
| | 134 | Condition Flags | |
| | 135 | Conditional Execute Structure | |

**Encoding: ELEMENT_TYPE**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| | 136 | Conditional Include Structure | |
| | 137 | Conditional Return | |
| | 138 | Include Structure | |
| | 139 | Test Extent | |
| | 140 | Test Extent 2D | |
| | 141 | Texture Mapping Sample Frequency | |
| | 142 | Surface Active Textures | |
| | 143 | Backface Surface Active Textures | |
| | 144 | Texture Mapping Perspective Correction | |
| | 145 | Texture Mapping Resource Hints | |
| | 146 | Polyline Set with Data (V5.2) | |
| | 147 | Fill Area Set with Data (V5.2) | |
| | 148 | Set of Fill Area Sets with Data (V5.2) | |
| | 149 | Triangle Strip with Data (V5.2) | |
| | 150 | Quadrilateral Mesh with Data (V5.2) | |
| | 151 | Global Transform with Hints | |
| | 152 | Global Transform 2D with Hints | |
| | 153 | Local Transform with Hints | |
| | 154 | Local Transform 2D with Hints | |
| | 155-0x7FFF | reserved for registration | |
| | 0x8000-0x80FF | reserved | |
| | 0x8100-0xFEFF | vendor OCs | |
| | 0xFF00-0xFFFE | reserved | |
| | 0xFFFF | *OCNil*†1 | |

†1) All elements are of type *OCAll,* including *OCNil*. *OCNil* is the type of the implicit element at element position 0.

### 3.7.7.50 ENUM_TYPE_INDEX

ENUM_TYPE_INDEX : INT16

### 3.7.7.51 FLOAT

FLOAT : floating-point value (See §3.7.5 for the encodings.)

### 3.7.7.52 FLOAT_FORMAT

FLOAT_FORMAT : ENUM_TYPE_INDEX (See §3.7.5 for the defined values of *FloatFormat*.)

### 3.7.7.53 FONT_ID

FONT_ID : {PEX_FONT_ID, X11_FONT_ID}

### 3.7.7.54 FP_DATA_COUNTS

FP_DATA_COUNTS : [facet_fp_count, vertex_fp_count : CARD16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | number of floats in the facet floating-point data list |
| 2 | CARD16 | number of floats in the vertex floating-point data list |

### 3.7.7.55 HALF_SPACE

HALF_SPACE : [point : COORD_3D,
vector : VECTOR_3D]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COORD_3D | point |
| 3**fp* | VECTOR_3D | vector |

### 3.7.7.56 HALF_SPACE_2D

HALF_SPACE_2D : [point : COORD_2D,
vector : VECTOR_2D]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2**fp* | COORD_2D | point |
| 2**fp* | VECTOR_2D | vector |

### 3.7.7.57 HATCH_STYLE

HATCH_STYLE : ENUM_TYPE_INDEX (See Table 6-42 for the defined values of *HatchStyle*.)

### 3.7.7.58 HIGHLIGHT_METHOD

HIGHLIGHT_METHOD : ENUM_TYPE_INDEX (See Table 7-4 for the defined values of *HighlightMethod*.)

### 3.7.7.59 HLHSR_IDENTIFIER

HLHSR_IDENTIFIER : INT32 (an expansion of the *HLHSRIdentifier* enumerated type values, see §6.9.38)

### 3.7.7.60 HLHSR_MODE

HLHSR_MODE : ENUM_TYPE_INDEX (See §10.2.20 for the defined values of *HLHSRMode*.)

### 3.7.7.61 HOMOGENEOUS_COLOR

HOMOGENEOUS_COLOR: {HOMO_COLOR_RGB_FLOAT, HOMO_COLOR_CIE_FLOAT, HOMO_COLOR_HSV_FLOAT, HOMO_COLOR_HLS_FLOAT, HOMO_COLOR_RGB_INT8, HOMO_COLOR_RGB_INT16}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *hsc*†1 | | homogeneous color value |

†1) *hsc* is 4 if the color type is *RGBInt8*. *hsc* is 8 if the color type is *RGBInt16*. *hsc* is 4**fp* if the color type is *RGBFloat*, *CIEFloat*, *HSVFloat*, or *HLSFloat*.

### 3.7.7.62 HOMO_COLOR_CIE_FLOAT

HOMO_COLOR_CIE_FLOAT: [color : COLOR_CIE_FLOAT,
w : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COLOR_CIE_FLOAT | color†1 |
| *fp* | FLOAT | w |

†1) The color components are divided by the homogeneous coordinate w before they are used.

### 3.7.7.63 HOMO_COLOR_HLS_FLOAT

HOMO_COLOR_HLS_FLOAT: [color : COLOR_HLS_FLOAT,
w : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COLOR_HLS_FLOAT | color†1 |
| *fp* | FLOAT | w |

†1) The color components are divided by the homogeneous coordinate w before they are used.

### 3.7.7.64 HOMO_COLOR_HSV_FLOAT

HOMO_COLOR_HSV_FLOAT: [color : COLOR_HSV_FLOAT,
w : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COLOR_HSV_FLOAT | color†1 |
| *fp* | FLOAT | w |

†1) The color components are divided by the homogeneous coordinate w before they are used.

### 3.7.7.65 HOMO_COLOR_RGB_FLOAT

HOMO_COLOR_RGB_FLOAT: [color : COLOR_RGB_FLOAT,
w : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COLOR_RGB_FLOAT | color†1 |
| *fp* | FLOAT | w |

†1) The color components are divided by the homogeneous coordinate w before they are used.

### 3.7.7.66 HOMO_COLOR_RGB_INT8

HOMO_COLOR_RGB_INT8: [red, green, blue, w : INT8]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | INT8 | red†1 |
| 1 | INT8 | green |
| 1 | INT8 | blue |
| 1 | INT8 | w |

†1) The red, green, and blue components are divided by the homogeneous coordinate w before they are used.

### 3.7.7.67 HOMO_COLOR_RGB_INT16

HOMO_COLOR_RGB_INT16: [red, green, blue, w : INT16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | INT16 | red†1 |
| 2 | INT16 | green |
| 2 | INT16 | blue |
| 2 | INT16 | w |

†1) The red, green, and blue components are divided by the homogeneous coordinate w before they are used.

### 3.7.7.68 INDEXED_COLOR

INDEXED_COLOR : [index : TABLE_INDEX]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | TABLE_INDEX | index |
| 2 | | unused |

#### 3.7.7.69 INTERIOR_STYLE

INTERIOR_STYLE : ENUM_TYPE_INDEX (See §6.9.84 for the defined values of *InteriorStyle*.)

#### 3.7.7.70 INT8

INT8 : signed 8-bit integer

#### 3.7.7.71 INT16

INT16 : signed 16-bit integer

#### 3.7.7.72 INT32

INT32 : signed 32-bit integer

#### 3.7.7.73 LIGHT_TYPE

LIGHT_TYPE : ENUM_TYPE_INDEX (See §7.2.10 for the defined values of *LightType*.)

#### 3.7.7.74 LINE_CAP_STYLE

LINE_CAP_STYLE : ENUM_TYPE_INDEX (See §6.9.43 for the defined values of *LineCapStyle*.)

#### 3.7.7.75 LINE_JOIN_STYLE

LINE_JOIN_STYLE : ENUM_TYPE_INDEX (See §6.9.46 for the defined values of *LineJoinStyle*.)

#### 3.7.7.76 LINE_TYPE

LINE_TYPE : ENUM_TYPE_INDEX (See §6.9.48 for the defined values of *LineType*.)

#### 3.7.7.77 LOOKUP_TABLE_ID

LOOKUP_TABLE_ID : RESOURCE_ID

#### 3.7.7.78 MARKER_TYPE

MARKER_TYPE : TYPE_OR_TABLE_INDEX (See §6.9.58 for the defined values of *MarkerType*.)

#### 3.7.7.79 MATRIX

MATRIX : FLOAT[4][4]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 16**fp*†1 | FLOAT | matrix |

†1) The matrix is encoded as a one-dimensional array of floating-point values. In the 4x4 array, the fourth element of the array represents the *x*-translation value, the eighth element the *y*-translation value, and so on.

### 3.7.7.80 MATRIX_3X3

MATRIX_3X3 : FLOAT[3][3]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 9*$fp$†1 | FLOAT | matrix |

†1) The matrix is encoded as a one-dimensional array of floating-point values. In the 3x3 array, the third element of the array represents the $x$-translation value, the sixth element the $y$-translation value, and so on.

### 3.7.7.81 MATRIX_HINTS

MATRIX_HINTS : BITMASK

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BITMASK | matrix hints |
| | 0x00000000 *none* | |
| | 0x00000001 *identity* | |
| | 0x00000002 *translation* | |
| | 0x00000004 *rotation* | |
| | 0x00000008 *reflection* | |
| | 0x00000010 *uniform_scale* | |
| | 0x00000020 *nonuniform_scale* | |
| | 0x00000040 *shear* | |
| | 0x00000080 *perspective_projective* | |
| | 0x00000100 *nonperspective_projective* | |
| | 0x00000200 *3D_to_2D_parallel_proj* | |
| | 0x00000400 *3D_to_2D_perspective_proj* | |
| | 0x00000800 *zero* | |
| | 0x00001000 *singular_and_unknown_type* | |
| | 0x00002000 *nonsingular_and_un-known_type* | |

### 3.7.7.82 MODEL_CLIP_OPERATOR

MODEL_CLIP_OPERATOR: ENUM_TYPE_INDEX (See §6.9.61 for the defined values of *ModelClipOperator*.)

### 3.7.7.83 MODEL_MATRIX_COMPOSITION

MODEL_MATRIX_COMPOSITION :
{*PreConcatenate*, *PostConcatenate*, *Replace*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | model matrix composition type |
| | 0 | *PreConcatenate* 5.0 | |
| | 1 | *PostConcatenate* 5.0 | |
| | 2 | *Replace* 5.0 | |

### 3.7.7.84 MONO_ENCODING

MONO_ENCODING : [char_set : CARD16,
char_set_width : {*csByte*, *csShort*, *csLong*},
encoding_state : CARD8,
string : LISTofCHARACTER]

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | character set |
| 1 | CARD8 | | character set width |
| | 0 | *csByte* 5.0 | |
| | 1 | *csShort* 5.0 | |
| | 2 | *csLong* 5.0 | |
| 1 | CARD8 | | encoding state |
| 2 | | | unused |
| 2 | CARD16 | | number of characters ($c$) |
| $s^{\dagger 1}$ | LISTofCHARACTER | | string |
| $p$ | | | $p$=pad($s$) |

†1) $s = c$ if the character set width is *csByte*. $s = 2c$ if the character set width is *csShort*. $s = 4c$ if the character set width is *csLong*.

### 3.7.7.85 NAME

NAME : CARD32

### 3.7.7.86 NAME_SET_ID

NAME_SET_ID : RESOURCE_ID

### 3.7.7.87 NPC_SUBVOLUME

NPC_SUBVOLUME : [min, max : COORD_3D]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COORD_3D | minimum |
| 3**fp* | COORD_3D | maximum |

### 3.7.7.88 OPT_ANGLES

OPT_ANGLES : optional [start, stop : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | start angle (radians) |
| *fp* | FLOAT | stop angle (radians) |

### 3.7.7.89 OPT_AXES

OPT_AXES : optional
[major_x, major_y : FLOAT,
major_z : OPT_FLOAT,
minor_x, minor_y : FLOAT,
minor_z : OPT_FLOAT]

**Encoding:**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | major axis x |
| *fp* | FLOAT | major axis y |
| *fp* | OPT_FLOAT | major axis z |
| *fp* | FLOAT | minor axis x |
| *fp* | FLOAT | minor axis y |
| *fp* | OPT_FLOAT | minor axis z |

†1) Let *x* be the size of an OPT_AXES in bytes. The base size is 4**fp*. The size increases by 2**fp* if the *2D_coords* vertex attribute bit is **not** set, indicating 3D coordinates are instead present.

### 3.7.7.90 OPT_BITMASK

OPT_BITMASK : optional BITMASK

### 3.7.7.91 OPT_COLOR

OPT_COLOR : optional COLOR

### 3.7.7.92 OPT_DATA

OPT_DATA : [color : OPT_COLOR,
normal : OPT_NORMAL,
edges : OPT_BITMASK,
radius : OPT_FLOAT,
angles : OPT_ANGLES,
axes : OPT_AXES,
fp_data : OPT_LISTofFLOAT]

**Encoding:**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| *sc* | OPT_COLOR | color (See COLOR for *sc*) |
| 3**fp* | OPT_NORMAL | normal |
| 4 | OPT_BITMASK | edges |
| *fp* | OPT_FLOAT | radius |
| 2**fp* | OPT_ANGLES | start and stop angles |
| *x* | OPT_AXES | major and minor axes (See OPT_AXES for *x*) |
| *n*fp* | OPT_LISTofFLOAT | list of floating-point data |

†1) All fields are optionally specified. Size *d* of an OPT_DATA is determined as follows:

If the primitive is an Arc Set with Data or Arc Area Set with Data, the radius is present and the base size is *fp* if the arc type is *Circle,* the radius and angles are present and the base size is 3**fp* if the arc type is *CircularArc*, *CircularClosedPieArc*, or *CircularClosedChordArc*, the axes are present and the base size is *x* if the arc type is *Ellipse*, and the angles and axes are present and the base size is *x*+2**fp* if the arc type is *EllipticalArc*, *EllipticalClosedPieArc*, or *EllipticalClosedChordArc*.

If the primitive is a Triangle Strip Fan Set with Data, the edges bitmask is present and the base size is 4.

For all other primitives the base size is 0.

The size increases by *sc* if the *color* vertex attribute is set, by 3**fp* if the *normal* vertex attribute is set, by 4 if the *edges* vertex attribute is set, by *n*fp* if the *fp_data* vertex attribute is set (where *n* is determined by a variable of type FP_DATA_COUNTS in the primitive's encoding), and by *fp* if the *radius* vertex attributes bit is set.

### 3.7.7.93 OPT_FLOAT

OPT_FLOAT : optional FLOAT

### 3.7.7.94 OPT_LISTofFLOAT

OPT_LISTofFLOAT : optional LISTofFLOAT

### 3.7.7.95 OPT_NORMAL

OPT_NORMAL : optional VECTOR_3D

### 3.7.7.96 OPT_POINT

OPT_POINT : optional COORD_3D

### 3.7.7.97 OUTPUT_CMD

OUTPUT_CMD : [element_type : ELEMENT_TYPE,
size : CARD16,
data : data]

The encoding for *data* is identical to the encoding used to create each output command, except the 4-byte OC header is **not** included.

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | ELEMENT_TYPE | element type |
| 2 | CARD16 | 16-bit size ($n$+1) OR 0 |
| $k^{\dagger 1}$ | CARD32 | 32-bit size ($n$+2) |
| $4*n^{\dagger 2}$ | | output command data |

†1) $k$ = ((size == 0) ? 4 : 0)
†2) Since the output command encodings are all 4-byte aligned, there is no need to pad here.

### 3.7.7.98 PC_BITMASK

PC_BITMASK : BITMASK[3]

**Encoding: PC_BITMASK**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BITMASK | pipeline context attributes (word 0) |
| | 0x00000001 *marker_type* 5.0 | |
| | 0x00000002 *marker_scale_factor* 5.0 | |
| | 0x00000004 *marker_color* 5.0 | |
| | 0x00000008 *marker_bundle_index* 5.0 | |
| | 0x00000010 *text_font_index* 5.0 | |
| | 0x00000020 *text_precision* 5.0 | |
| | 0x00000040 *char_expansion* 5.0 | |
| | 0x00000080 *char_spacing* 5.0 | |
| | 0x00000100 *text_color* 5.0 | |
| | 0x00000200 *char_height* 5.0 | |
| | 0x00000400 *char_up_vector* 5.0 | |
| | 0x00000800 *text_path* 5.0 | |
| | 0x00001000 *text_alignment* 5.0 | |
| | 0x00002000 *annotation_text_height* 5.0 | |
| | 0x00004000 *annotation_text_up_vector* 5.0 | |
| | 0x00008000 *annotation_text_path* 5.0 | |
| | 0x00010000 *annotation_text_alignment* 5.0 | |
| | 0x00020000 *annotation_text_style* 5.0 | |
| | 0x00040000 *text_bundle_index* 5.0 | |
| | 0x00080000 *line_type* 5.0 | |
| | 0x00100000 *line_width* 5.0 | |
| | 0x00200000 *line_color* 5.0 | |
| | 0x00400000 *curve_approx_method* 5.0 | |
| | 0x00800000 *polyline_interpolation_method* 5.0 | |
| | 0x01000000 *line_bundle_index* 5.0 | |
| | 0x02000000 *interior_style* 5.0 | |
| | 0x04000000 *interior_style_index* 5.0 | |
| | 0x08000000 *surface_color* 5.0 | |
| | 0x10000000 *reflection_attributes* 5.0 | |
| | 0x20000000 *reflection_model* 5.0 | |
| | 0x40000000 *surface_interpolation_method* 5.0 | |
| | 0x80000000 *bf_interior_style* 5.0 | |

**Encoding: PC_BITMASK**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BITMASK | (word 1) |
| | 0x00000001 *bf_interior_style_index* 5.0 | |
| | 0x00000002 *bf_surface_color* 5.0 | |
| | 0x00000004 *bf_reflection_attributes* 5.0 | |
| | 0x00000008 *bf_reflection_model* 5.0 | |
| | 0x00000010 *bf_surface_interpolation_method* 5.0 | |
| | 0x00000020 *surface_approx_method* 5.0 | |
| | 0x00000040 *facet_culling_mode* 5.0 | |
| | 0x00000080 *facet_distinguish_flag* 5.0 | |
| | 0x00000100 *pattern_size* 5.0 | |
| | 0x00000200 *pattern_reference_point* 5.0 | |
| | 0x00000400 *pattern_reference_vector1* 5.0 | |
| | 0x00000800 *pattern_reference_vector2* 5.0 | |
| | 0x00001000 *interior_bundle_index* 5.0 | |
| | 0x00002000 *surface_edge_flag* 5.0 | |
| | 0x00004000 *surface_edge_type* 5.0 | |
| | 0x00008000 *surface_edge_width* 5.0 | |
| | 0x00010000 *surface_edge_color* 5.0 | |
| | 0x00020000 *edge_bundle_index* 5.0 | |
| | 0x00040000 *local_transform* 5.0 | |
| | 0x00080000 *global_transform* 5.0 | |
| | 0x00100000 *model_clip_indicator* 5.0 | |
| | 0x00200000 *model_clip_volume* 5.0 | |
| | 0x00400000 *view_index* 5.0 | |
| | 0x00800000 *light_source_state* 5.0 | |
| | 0x01000000 *depth_cue_index* 5.0 | |
| | 0x02000000 *aspect_source_flags* 5.0 | |
| | 0x04000000 *pick_id* 5.0 | |
| | 0x08000000 *HLHSR_identifier* 5.0 | |
| | 0x10000000 *name_set* 5.0 | |
| | 0x20000000 *color_approx_index* 5.0 | |
| | 0x40000000 *rendering_color_model* 5.0 | |
| | 0x80000000 *parametric_surface_characteristics* 5.0 | |

**Encoding: PC_BITMASK**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BITMASK | (word 2) |
| | 0x00000001 *alpha_blend_function* 5.2 | |
| | 0x00000002 *bf_data_map_index* 5.2 | |
| | 0x00000004 *bf_data_map_method* 5.2 | |
| | 0x00000008 *bf_interior_bundle_index* 5.2 | |
| | 0x00000010 *bf_reflectance_bundle_index* 5.2 | |
| | 0x00000020 *condition_flags* 5.2 | |
| | 0x00000040 *data_map_index* 5.2 | |
| | 0x00000080 *data_map_method* 5.2 | |
| | 0x00000100 *drawing_function* 5.2 | |
| | 0x00000200 *more_aspect_source_flags* 5.2 | |
| | 0x00000400 *global_transform_hints* 5.2 | |
| | 0x00000800 *highlight_index* 5.2 | |
| | 0x00001000 *line_cap_style* 5.2 | |
| | 0x00002000 *line_join_style* 5.2 | |
| | 0x00004000 *line_miter_limit* 5.2 | |
| | 0x00008000 *local_transform_hints* 5.2 | |
| | 0x00010000 *parametric_surface_bundle_index* 5.2 | |
| | 0x00020000 *plane_mask* 5.2 | |
| | 0x00040000 *primitive_antialiasing* 5.2 | |
| | 0x00080000 *reflectance_bundle_index* 5.2 | |
| | 0x00100000 *sequential_processing* 5.2 | |
| | 0x00200000 *vertex_order* 5.2 | |
| | 0x00400000 *Zbuffer_test* 5.2 | |
| | 0x00800000 *active_textures* 5.2 | |
| | 0x01000000 *bf_active_textures* 5.2 | |
| | 0x02000000 *tm_perspective_correction* 5.2 | |
| | 0x04000000 *tm_resource_hints* 5.2 | |
| | 0x08000000 *tm_sample_frequency* 5.2 | |

### 3.7.7.99 PEX_FONT_ID

PEX_FONT_ID : RESOURCE_ID

### 3.7.7.100 PICK_ALL_METHOD

PICK_ALL_METHOD : ENUM_TYPE_INDEX (See §11.1.3.2 for the defined values of *PickAllMethod*.)

### 3.7.7.101 PICK_DATA

PICK_DATA : {DC_HIT_BOX_DATA (§3.7.7.38), NPC_SUBVOLUME (§3.7.7.87)}

### 3.7.7.102 PICK_DEVICE_TYPE

PICK_DEVICE_TYPE : ENUM_TYPE_INDEX (See §11.1.2 for the defined values of *PickDeviceType*.)

### 3.7.7.103 PICK_ELEMENT_REF

PICK_ELEMENT_REF : [s_id : STRUCTURE_ID,
offset, pickid : CARD32]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | STRUCTURE_ID | structure identifier |
| 4 | CARD32 | offset |
| 4 | CARD32 | pick identifier |

### 3.7.7.104 PICK_ONE_METHOD

PICK_ONE_METHOD : ENUM_TYPE_INDEX (See §11.1.3.1 for the defined values of *PickOneMethod*.)

### 3.7.7.105 PICK_RECORD

PICK_RECORD : [pick_type : PICK_DEVICE_TYPE,
hit_box : PICK_DATA]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | PICK_DEVICE_TYPE | pick type (§3.7.7.102) |
| 2 | | unused |
| $n^{\dagger 1}$ | PICK_DATA | hit box |

†1) *n* is 4+*fp* if the pick type is *DCHitBox*. *n* is 6**fp* if the pick type is *NPCHitVolume*.

### 3.7.7.106 PIPELINE_CONTEXT_ID

PIPELINE_CONTEXT_ID : RESOURCE_ID

### 3.7.7.107 PIXMAP_ID

PIXMAP_ID : RESOURCE_ID

### 3.7.7.108 POLYLINE_INTERP

POLYLINE_INTERP : ENUM_TYPE_INDEX (See §6.9.68 for the defined values of *PolylineInterpMethod*.)

### 3.7.7.109 PRIM_AA

PRIM_AA : [prims : PRIM_AA_BITMASK,
function : PRIM_AA_BLEND_FUNC]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | PRIM_AA_BITMASK | antialiased primitives types |
| 2 | PRIM_AA_BLEND_FUNC | primitive antialiasing blend function |

### 3.7.7.110 PRIM_AA_BLEND_FUNC

PRIM_AA_BLEND_FUNC : ENUM_TYPE_INDEX (See §6.9.69 for the defined values of *PrimAntiAliasBlendFunction*.)

### 3.7.7.111 PRIM_AA_BITMASK

PRIM_AA_BITMASK : BITMASK_SHORT

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | BITMASK_SHORT | antialiased primitive types |
| | 0x0001 *point* 5.2 | |
| | 0x0002 *vector* 5.2 | |
| | 0x0004 *polygon* 5.2 | |
| | 0x0008 *raster_text* 5.2 | |
| | 0x0010 *vector_text* 5.2 | |
| | 0x0020 *polygonal_text* 5.2 | |

### 3.7.7.112 PRIM_DATA_ATTRIBUTES

PRIM_DATA_ATTRIBUTES: BITMASK_SHORT

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | BITMASK_SHORT[†1] | primitive data attributes |
| | 0x0001 *color* 5.0 | |
| | 0x0002 *normal* 5.0 | |
| | 0x0004 *edges* 5.0 | |
| | 0x0008 *fp_data* 5.2 | |
| | 0x0010 *radius* 5.2 | |
| | 0x0020 *2D_coords* 5.2 | |

†1) Multiple bits may be set. Each bit indicates optional data for the primitive. Each primitive output command using type PRIM_DATA_ATTRIBUTES defines which bits may be set for that primitive and the interpretation of each bit.

### 3.7.7.113 PSC_TYPE

PSC_TYPE : ENUM_TYPE_INDEX (See §6.9.63 for the defined values of *ParametricSurfaceCharacteristics*.)

### 3.7.7.114 PSURF_CHAR

PSURF_CHAR : [psc_type : PSC_TYPE,
psc_data : data]

The encoding of *psc_data* depends on the parametric surface characteristics type, *psc_type*. The encoding of *psc_data* for each value of *psc_type* immediately follows the encoding of PSURF_CHAR.

**Encoding:**

| # of Bytes[†1] | Data Type or Value | Description |
|---|---|---|
| 2 | PSC_TYPE | parametric surface characteristics type |
| 2 | CARD16 | size of psc_data (*n*) |
| *n* | | parametric surface characteristics data |
| *p* | | pad(*n*) |

†1) Size *d* of a PSURF_CHAR is 4+*n*+*p*. See the encoding for each parametric surface characteristics type below for *n*.

PSC_TYPE = *None* data : none

PSC_TYPE = *ImpDep* data : none

PSC_TYPE = *IsoparametricCurves* data :
[placement : {*Uniform, NonUniform*},
num_u_curves, num_v_curves : CARD16]

**Encoding: psc_data for PSC_TYPE *IsoparametricCurves***

| # of Bytes[†1] | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | placement type |
| | 0 *Uniform* 5.0 | |
| | 1 *NonUniform* 5.0 | |
| 2 | | unused |
| 2 | CARD16 | number of curves in u direction |
| 2 | CARD16 | number of curves in v direction |

†1) Size *n* of the data for parametric surface characteristics type *IsoparametricCurves* is 8.

PSC_TYPE = *MC_LevelCurves* data :
[MC_origin : COORD_3D,
MC_direction : VECTOR_3D,
num_intersections : CARD16,
intersection_params : LISTofFLOAT]

**Encoding: psc_data for PSC_TYPE *MC_LevelCurves***

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COORD_3D | MC origin |
| 3**fp* | VECTOR_3D | MC direction |
| 2 | CARD16 | number of intersections (*t*) |
| 2 | | unused |
| *t*fp* | LISTofFLOAT | intersection parameters |

†1) Size *n* of the data for parametric surface characteristics type *MC_LevelCurves* is 4+(6+*t*)**fp*.

PSC_TYPE = *WC_LevelCurves* data :
[WC_origin : COORD_3D,
WC_direction : VECTOR_3D,
num_intersections : CARD16,
intersection_params : LISTofFLOAT]

**Encoding: psc_data for PSC_TYPE *WC_LevelCurves***

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| 3**fp* | COORD_3D | WC origin |
| 3**fp* | VECTOR_3D | WC direction |
| 2 | CARD16 | number of intersections (*t*) |
| 2 | | unused |
| *t*fp* | LISTofFLOAT | intersection parameters |

†1) Size *n* of the data for parametric surface characteristics type *WC_LevelCurves* is 4+(6+*t*)**fp*.

### 3.7.7.115 REFLECTION_ATTR

REFLECTION_ATTR : [ambient_coef, diffuse_coef, specular_coef : FLOAT,
specular_conc, transmission_coef : FLOAT,
specular_color : COLOR_SPECIFIER]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | ambient coefficient |
| *fp* | FLOAT | diffuse coefficient |
| *fp* | FLOAT | specular coefficient |
| *fp* | FLOAT | specular concentration |
| *fp* | FLOAT | transmission coefficient |
| 4+*sc* | COLOR_SPECIFIER | specular color (see COLOR for *sc*) |

### 3.7.7.116 REFLECTION_MODEL

REFLECTION_MODEL : ENUM_TYPE_INDEX (See §6.9.88 for the defined values of *ReflectionModel*.)

### 3.7.7.117 RELATIONAL_TEST

RELATIONAL_TEST : {*LessThan, LessThanOrEqual, GreaterThan, GreaterThanOrEqual, Equal, NotEqual*, *Pass*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | relational test |
| | 0 *LessThan* | |
| | 1 *LessThanOrEqual* | |
| | 2 *GreaterThan* | |
| | 3 *GreaterThanOrEqual* | |
| | 4 *Equal* | |
| | 5 *NotEqual* | |
| | 6 *Pass* | |

### 3.7.7.118 RENDERER_ID

RENDERER_ID : RESOURCE_ID

### 3.7.7.119 RENDERING_COLOR_MODEL

RENDERING_COLOR_MODEL:ENUM_TYPE_INDEX (See §6.9.72 for the defined values of *RenderingColorModel*.)

### 3.7.7.120 RESOURCE_ID

RESOURCE_ID : 32-bit identifier

In PEX, when a request is made to free a resource, the request deletes the association between the resource ID and the resource. However the resource's storage is not freed until no other resource references it. For a resource's attribute whose type is RESOURCE_ID, the attribute's value is one of the following: a valid resource ID, the PEX constant *AlreadyFreed* (Table 3-3 ) if the corresponding resource has been freed, or the X constant **None,** the *Null* resource identifier, if the attribute doesn't reference a resource.

**Table 3-3 . PEX Freed Resource ID Constant Value**

| Value | Symbol | Description |
|---|---|---|
| 1 | *AlreadyFreed* | The association between a resource ID and its resource has been deleted, but some other resource still references the resource ID. |

### 3.7.7.121 SEARCH_CONTEXT_ID

SEARCH_CONTEXT_ID : RESOURCE_ID

### 3.7.7.122 SHAPE

SHAPE : {*Complex*, *Nonconvex*, *Convex*, *Unknown*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | shape hint |
| | 0 | *Complex* 5.0 | |
| | 1 | *Nonconvex* 5.0 | |
| | 2 | *Convex* 5.0 | |
| | 3 | *Unknown* 5.0 | |

### 3.7.7.123 SPLINE_COLOR

SPLINE_COLOR : {DIRECT_COLOR, HOMOGENEOUS_COLOR}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| $c^{\dagger 1}$ | | spline color value |

†1) *c* is *sc* if the color is nonhomogeneous. *c* is *hsc* if the color is homogeneous.

### 3.7.7.124 STRING

STRING : {STRING_ALIGNED_PACKED, STRING_UNALIGNED_PACKED, STRING_ALIGNED_PADDED, STRING32}

### 3.7.7.125 STRING_ALIGNED_PACKED

STRING_ALIGNED_PACKED :
LISTofCARD8

In the encoding, the *length* field of STRING_ALIGNED_PACKED always starts on a 4-byte boundary.

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | length (*n*) |
| *n* | LISTofCARD8 | string |
| *p* | | pad(*n*+2) |

### 3.7.7.126 STRING_ALIGNED_PADDED

STRING_ALIGNED_PADDED :
LISTofCARD8

In the encoding, the *length* field of STRING_ALIGNED_PADDED always starts on a 4-byte boundary.

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | length (*n*) |
| 2 | | unused |
| *n* | LISTofCARD8 | string |
| *p* | | pad(*n*) |

### 3.7.7.127 STRING_UNALIGNED_PACKED

STRING_UNALIGNED_PACKED :
LISTofCARD8

In the encoding, the *length* field of STRING_UNALIGNED_PACKED always starts two bytes beyond a 4-byte boundary.

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | length (*n*) |
| *n* | LISTofCARD8 | string |
| *p* | | pad(*n*) |

### 3.7.7.128 STRING32

STRING32 : LISTofCARD8

In the encoding, the *length* field of STRING32 always starts on a 4-byte boundary.

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | length (*n*) |
| *n* | LISTofCARD8 | string |
| *p* | | pad(*n*) |

### 3.7.7.129 STRUCTURE_ID

STRUCTURE_ID : RESOURCE_ID

### 3.7.7.130 SURFACE_APPROX

SURFACE_APPROX : [approx_method : SURFACE_APPROX_METHOD,
u_tolerance, v_tolerance : FLOAT]

**Encoding: SURFACE_APPROX**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | SURFACE_APPROX_METHOD | surface approximation method |
| 2 | | unused |
| *fp* | FLOAT | approximation *u* tolerance |
| *fp* | FLOAT | approximation *v* tolerance |

### 3.7.7.131 SURFACE_APPROX_METHOD

SURFACE_APPROX_METHOD:
ENUM_TYPE_INDEX (See §6.9.75 for the defined values of *SurfaceApproxMethod*.)

### 3.7.7.132 SURFACE_EDGE_TYPE

SURFACE_EDGE_TYPE : ENUM_TYPE_INDEX (See §6.9.82 for the defined values of *SurfaceEdgeType*.)

### 3.7.7.133 SURFACE_INTERP

SURFACE_INTERP : ENUM_TYPE_INDEX (See §6.9.86 for the defined values of *SurfaceInterpMethod*.)

### 3.7.7.134 SWITCH

SWITCH : {*Off, On*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | switch |

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| | 0 | *Off* 5.0 | |
| | 1 | *On* 5.0 | |

### 3.7.7.135 TABLE_BITMASK

TABLE_BITMASK : {COLOR_TABLE_BITMASK, COLOR_APPROX_TABLE_BITMASK, EXTENT_SIZE_TABLE_BITMASK, DATA_MAP_TABLE_BITMASK, DEPTH_CUE_TABLE_BITMASK, EDGE_TABLE_BITMASK, HIGHLIGHT_TABLE_BITMASK, INTERIOR_TABLE_BITMASK, LIGHT_TABLE_BITMASK, LINE_TABLE_BITMASK, MARKER_TABLE_BITMASK, MARKER_GLYPH_TABLE_BITMASK, PATTERN_TABLE_BITMASK, TEXT_TABLE_BITMASK, TEXT_FONT_TABLE_BITMASK, TEXTURE_MAP_TABLE_BITMASK, VIEW_TABLE_BITMASK}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BITMASK†1 | table bitmask |

†1) The definition of the bitmask for each type of table is shown with the table's definition in Chapter 7.

### 3.7.7.136 TABLE_ENTRY

TABLE_ENTRY : {COLOR_TABLE_ENTRY, COLOR_APPROX_TABLE_ENTRY, DEPTH_CUE_TABLE_ENTRY, EDGE_TABLE_ENTRY, INTERIOR_TABLE_ENTRY, LIGHT_TABLE_ENTRY, LINE_TABLE_ENTRY, MARKER_TABLE_ENTRY, PATTERN_TABLE_ENTRY, TEXT_TABLE_ENTRY, TEXT_FONT_TABLE_ENTRY, VIEW_TABLE_ENTRY}

Type TABLE_ENTRY is used by requests which are not recommended for use by clients using PEX version 5.2 or later. It only supports table types defined prior to version 5.2.

The definition of the entry for each type of table is shown with the table's definition in Chapter 7.

### 3.7.7.137 TABLE_ENTRY_STATUS

TABLE_ENTRY_STATUS : {*Default, Defined*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | table entry status |
| | 0 *Default* 5.0 | |
| | 1 *Defined* 5.0 | |

### 3.7.7.138 TABLE_INDEX

TABLE_INDEX : CARD16

### 3.7.7.139 TABLE_TYPE

TABLE_TYPE : {*LineBundle*, *MarkerBundle*, *TextBundle*, *InteriorBundle, EdgeBundle, Pattern*, *TextFont*, *Color*, *View*, *Light*, *DepthCue*, *ColorApprox, ExtentSize, DataMapping, Highlight, MarkerGlyph, TextureMapping*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | table type |
| | 1 *LineBundle* 5.0 | |
| | 2 *MarkerBundle* 5.0 | |
| | 3 *TextBundle* 5.0 | |
| | 4 *InteriorBundle* 5.0 | |
| | 5 *EdgeBundle* 5.0 | |
| | 6 *Pattern* 5.0 | |
| | 7 *TextFont* 5.0 | |
| | 8 *Color* 5.0 | |
| | 9 *View* 5.0 | |
| | 10 *Light* 5.0 | |
| | 11 *DepthCue* 5.0 | |
| | 12 *ColorApprox* 5.0 | |
| | 13 *ExtentSize* 5.2 | |
| | 14 *DataMapping* 5.2 | |
| | 15 *Highlight* 5.2 | |
| | 16 *MarkerGlyph* 5.2 | |
| | 17 *TextureMapping* 5.2 | |

#### 3.7.7.140 TABLE_TYPE_5_1

TABLE_TYPE_5_1 : {*LineBundle*, *MarkerBundle*, *TextBundle*, *InteriorBundle, EdgeBundle, Pattern*, *TextFont*, *Color*, *View*, *Light*, *DepthCue*, *ColorApprox*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | table type |
| | 1 | *LineBundle* 5.0 | |
| | 2 | *MarkerBundle* 5.0 | |
| | 3 | *TextBundle* 5.0 | |
| | 4 | *InteriorBundle* 5.0 | |
| | 5 | *EdgeBundle* 5.0 | |
| | 6 | *Pattern* 5.0 | |
| | 7 | *TextFont* 5.0 | |
| | 8 | *Color* 5.0 | |
| | 9 | *View* 5.0 | |
| | 10 | *Light* 5.0 | |
| | 11 | *DepthCue* 5.0 | |
| | 12 | *ColorApprox* 5.0 | |

This type is used by requests which support only types of lookup tables which were defined prior to PEX version 5.2.

#### 3.7.7.141 TABLE_VALUE_TYPE

TABLE_VALUE_TYPE : {*Set, Realized*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | table value type |
| | 0 | *Set* 5.0 | |
| | 1 | *Realized* 5.0 | |

#### 3.7.7.142 TEXT_ALIGNMENT

TEXT_ALIGNMENT : [vertical : TEXT_VALIGNMENT, horizontal : TEXT_HALIGNMENT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | TEXT_VALIGNMENT | text vertical alignment |
| 2 | TEXT_HALIGNMENT | text horizontal alignment |

### 3.7.7.143 TEXT_HALIGNMENT

TEXT_HALIGNMENT : {*HAlignNormal*, *HAlignLeft*, *HAlignCenter, HAlignRight*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | text horizontal alignment |
| | 0 *HAlignNormal* 5.0 | |
| | 1 *HAlignLeft* 5.0 | |
| | 2 *HAlignCenter* 5.0 | |
| | 3 *HAlignRight* 5.0 | |

### 3.7.7.144 TEXT_PATH

TEXT_PATH : {*Right*, *Left*, *Up*, *Down*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | text path |
| | 0 *Right* 5.0 | |
| | 1 *Left* 5.0 | |
| | 2 *Up* 5.0 | |
| | 3 *Down* 5.0 | |

### 3.7.7.145 TEXT_PRECISION

TEXT_PRECISION : {*String*, *Char*, *Stroke*}

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | text precision |
| | 0 *String* 5.0 | |
| | 1 *Char* 5.0 | |
| | 2 *Stroke* 5.0 | |

#### 3.7.7.146 TEXT_VALIGNMENT

TEXT_VALIGNMENT : {*VAlignNormal*, *VAlignTop*, *VAlignCap*, *VAlignHalf*, *VAlignBase*, *VAlignBottom*}

**Encoding: TEXT_VALIGNMENT**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | text vertical alignment |
| | 0 *VAlignNormal* 5.0 | |
| | 1 *VAlignTop* 5.0 | |
| | 2 *VAlignCap* 5.0 | |
| | 3 *VAlignHalf* 5.0 | |
| | 4 *VAlignBase* 5.0 | |
| | 5 *VAlignBottom* 5.0 | |

#### 3.7.7.147 TEXTURE_MAP_ID

TEXTURE_MAP_ID : RESOURCE_ID

#### 3.7.7.148 TM_PERSPECTIVE_CORRECTION

TM_PERSPECTIVE_CORRECTION :
ENUM_TYPE_INDEX (See §6.9.96 for the defined values of *TMPerspectiveCorrection*.)

#### 3.7.7.149 TM_RESOURCE_HINTS

TM_RESOURCE_HINTS : ENUM_TYPE_INDEX (See §6.9.97 for the defined values of *TMResourceHints*.)

#### 3.7.7.150 TM_SAMPLE_FREQUENCY

TM_SAMPLE_FREQUENCY :
ENUM_TYPE_INDEX (See §6.9.98 for the defined values of *TMSampleFrequency*.)

#### 3.7.7.151 TRANSPARENCY_METHOD

TRANSPARENCY_METHOD :
ENUM_TYPE_INDEX (See §10.2.40 for defined values of *TransparencyMethod*.)

### 3.7.7.152 TRIM_CURVE

TRIM_CURVE : [visibility : SWITCH,
order : CARD16,
type : COORD_TYPE,
approx_method : TRIM_CURVE_APPROX_METHOD,
tolerance : FLOAT,
tmin, tmax : FLOAT
knots : LISTofFLOAT,
points : LISTofCOORD]

**Encoding: TRIM_CURVE**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| 1 | SWITCH | visibility |
| 1 | | unused |
| 2 | CARD16 | order |
| 2 | COORD_TYPE | curve type |
| 2 | TRIM_CURVE_APPROX_METHOD | approximation method |
| *fp* | FLOAT | approximation tolerance |
| *fp* | FLOAT | tmin |
| *fp* | FLOAT | tmax |
| 4 | CARD32 | number of knots (*k*) |
| 4 | CARD32 | number of control points (*p*) |
| *k***fp* | LISTofFLOAT | knots |
| *p***r*†2**fp* | LISTofCOORD†3 | control points |

†1) Size *c* of a TRIM_CURVE is 20+(3+*k*+*p***r*)**fp*.
†2) *r* is 2 if the curve is *NonRational*, else *r* is 3 because the curve is *Rational*.
†3) The control points are COORD_2D if *the curve* is *NonRational*, else they are COORD_3D because the *curve* is *Rational*.

### 3.7.7.153 TRIM_CURVE_APPROX_METHOD

TRIM_CURVE_APPROX_METHOD :
ENUM_TYPE_INDEX (See §6.7.24 for the defined values of *TrimCurveApproxMethod*.)

### 3.7.7.154 TYPE_OR_TABLE_INDEX

TYPE_OR_TABLE_INDEX : {ENUM_TYPE_INDEX, TABLE_INDEX}

#### 3.7.7.155 VECTOR_2D

VECTOR_2D : [x, y : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | x |
| *fp* | FLOAT | y |

#### 3.7.7.156 VECTOR_3D

VECTOR_3D : [x, y, z : FLOAT]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | x |
| *fp* | FLOAT | y |
| *fp* | FLOAT | z |

#### 3.7.7.157 VERTEX

VERTEX : [x, y : FLOAT,
z : OPT_FLOAT,
data : OPT_DATA]

**Encoding:**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | x |
| *fp* | FLOAT | y |
| *fp* | OPT_FLOAT | z |
| *d* | OPT_DATA | data (see OPT_DATA for *d*) |

†1) Size *v* of an VERTEX is determined as follows. The base size is 2**fp*+*d*. The size increases by *fp* if the vertex is a 3D vertex.

#### 3.7.7.158 VERTEX_ORDER

VERTEX_ORDER : {*Righthand, Lefthand*}

**Encoding:**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | vertex order |
| | 0 | *Righthand* | |
| | 1 | *Lefthand* | |

#### 3.7.7.159 VIEWPORT

VIEWPORT : [min : DEVICE_COORD
max : DEVICE_COORD,
use_drawable : BOOLEAN]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4+*fp* | DEVICE_COORD | minimum |
| 4+*fp* | DEVICE_COORD | maximum |
| 1 | BOOLEAN | use drawable |
| 3 | | unused |

#### 3.7.7.160 VISUALID

VISUALID : 32-bit visual identifier (top three bits guaranteed to be zero)

#### 3.7.7.161 WINDOW_ID

WINDOW_ID : RESOURCE_ID

#### 3.7.7.162 X11_FONT_ID

X11_FONT_ID : RESOURCE_ID

#### 3.7.7.163 Z_TEST

Z_TEST : ENUM_TYPE_INDEX (See §6.9.101 for defined values of *ZTest*.)

## 3.8 Enumerated Types

The PEX enumerations are defined where the request or functionality they apply to is defined. Table 4-3 (page 99) lists those locations.

Each enumerated type *Foo* is defined in a table labeled “Enumerated Type *Foo*”. The table defines the mnemonic and 16-bit numerical value for each value of the enumerated type. A description is also given for each value and a symbol representing the version of PEX when the value was introduced.

The enumeration type class and its values have an initial capital letter, use capital letters for separation, and are printed in italics (e.g., *FloatFormat*).

## 3.9 Constants

The PEX constants are defined in a variety of places. If a set of constants is used by only one request or output command (in some cases, two), it is defined in the encoding for that request or output command. If a set of constants is used by multiple requests or output commands, it is defined as a PEX type. If a description is required for the values within a set of constants, a table labeled “Foo Constant Values” is used, where Foo represents the name of the set of constants.

A constant has an initial capital letter, uses capital letters for separation, and is printed in italics (e.g., *NoClip*).

## 3.10 Errors

Errors are 32 bytes long. They include an 8-bit code identifying the reply as an error, an 8-bit error code identifying the specific error, the least significant 16 bits of the sequence number of the request that failed, a 32-bit value giving information pertaining to the failure, the major opcode identifying the extension reporting the error, and the minor opcode of the request in which the error was noted. The values contained in the unused bytes of an error are undefined.

The PEX Protocol uses the same set of error codes as the X11 Protocol when applicable. Additional error codes are provided for PEX-specific errors. The X11 errors generated by PEX requests are listed in Table 3-4 . Not all PEX requests exhaustively define which X11 errors are possible.

**Table 3-4 X11 Errors Generated by PEX Requests**

| Error Code | General Reason for the Error |
|---|---|
| *Alloc* | In the PEX context, this error is used as specified for X11. Note, as with X11 requests, the explicit listing of *Alloc* errors in PEX requests only covers allocation at a very coarse level and is not intended to (nor can it in practice hope to) cover all cases of a server running out of allocation space in the middle of service. 5.0 |
| *Drawable* | In the PEX context, this error is used as specified for X11. 5.0 |
| *IDChoice* | In the PEX context, this error is used as specified for X11. 5.0 |
| *Implementation* | In the PEX context, this error is used as specified for X11. 5.0 |
| *Length* | In the PEX context, this error is used as specified for X11. 5.0 |
| *Match* | In the PEX context, this error indicates a mismatch between the capabilities of the specified resource and the capabilities for which it is being used. 5.0 |
| *Pixmap* | In the PEX context, this error is used as specified for X11. 5.2 |
| *Request* | In the PEX context, this error is used as specified for X11. 5.0 |
| *Value* | In the PEX context, this error is used as specified for X11. 5.0 |

PEX errors are listed in Table 3-1 . Their encodings are given in Section 3.10.1.

If an error occurs while processing a request that modifies a PEX resource, it is possible that some portion of the resource or some of the resource’s attributes will have been altered.

**Table 3-5 . PEX Errors**

| Error Code | General Reason for the Error |
|---|---|
| *ColorType* | The specified color type is not supported. 5.0 |
| *FloatingPointFormat* | The specified floating point format is not supported. 5.0 |
| *Label* | The specified label does not exist in the structure. 5.0 |
| *LookupTable* | A lookup table ID does not name an existing lookup table. 5.0 |
| *NameSet* | A name set ID does not name an existing name set. 5.0 |
| *OutputCommand* | A value for some parameter of an output command is illegal, out of range, or otherwise inappropriate. 5.0 |
| *Path* | A value for a structure network path contains inappropriate or illegal values. 5.0 |
| *PEXFont* | A font ID does not name an existing PEX font resource. 5.0 |
| *PEXImpDep* | An implementation-dependent error has occurred. Additional information about the error may be included in the error reply; see your vendor product documentation. 5.2 |
| *PhigsWKS* | A PHIGS workstation argument is illegal or does not name a defined PHIGS workstation resource. 5.0 |
| *PickMeasure* | A pick measure argument is illegal or does not name a defined pick measure resource. 5.0 |
| *PipelineContext* | A pipeline context ID does not name an existing pipeline context. 5.0 |
| *Renderer* | A renderer ID does not name an existing renderer. 5.0 |
| *RendererState* | A renderer received a **PEXBeginRendering** request while in renderer state *Rendering*. 5.0 |
| *SearchContext* | A search context argument is illegal or does not name a defined search context resource. 5.0 |
| *Structure* | A structure ID does not name an existing structure. 5.0 |
| *StructurePermission* | An illegal operation was attempted on a structure whose permission is *StructureWriteOnly* or *StructureLocked*. 5.2 |

### 3.10.1 Error Encodings

**Encoding: ColorType Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 0 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | bad color type |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: FloatingPointFormat Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 2 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | bad floating-point format |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: Label Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 3 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | bad label |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: LookupTable Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 4 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | LOOKUP_TABLE_ID | bad lookup table ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: NameSet Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 5 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | NAME_SET_ID | bad name set ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: OutputCommand Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 14 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | RESOURCE_ID | renderer or structure ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 1 | | unused |
| 2 | CARD16 | opcode of bad output command |
| 2 | CARD16 | number of output commands in the request that were processed successfully |
| 16 | | unused |

**Encoding: Path Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 6 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | index of bad path element |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: PEXFont Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 7 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | FONT_ID | bad font ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: PEXImpDep Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 15 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | bad resource, enumerated type, or other information |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused, or information further explaining the error. |

**Encoding: PhigsWKS Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 8 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | PHIGS_WKS_ID | bad PHIGS workstation ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: PickMeasure Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 9 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | PICK_MEASURE_ID | bad device ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: PipelineContext Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 10 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | PIPELINE_CONTEXT_ID | bad pipeline context ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: Renderer Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 11 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | RENDERER_ID | bad renderer ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: RendererState Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 1 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | RENDERER_ID | renderer ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: SearchContext Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 12 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | SEARCH_CONTEXT_ID | bad search context ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: Structure Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 13 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | STRUCTURE_ID | bad structure ID |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

**Encoding: StructurePermission Error**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | 0 | Error |
| 1 | Base error code + 16 | PEX error code |
| 2 | CARD16 | sequence number |
| 4 | STRUCTURE_ID | ID of structure whose permission is inappropriate |
| 2 | CARD16 | minor opcode |
| 1 | CARD8 | major opcode |
| 21 | | unused |

## 3.11 Events

Events are 32 bytes long and contain an 8-bit event code indicating the event type, a 16-bit sequence number, and other event-type-specific information describing the event. The most-significant bit in the event code is set to 1 if the event is generated from a **SendEvent** request. The values contained in unused bytes of an event are undefined.

### 3.11.1 Event Encodings

**Encoding: MaxHitsReached Event** 5.1

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | Base event code + 0 | event code |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | RENDERER_ID | renderer ID |
| 24 | | event specific or unused |

# 4 General Information Requests

## 4.1 General Inquiry Requests

The requests defined in this chapter return information about the support provided by the PEX extension of a particular server. The requests are listed in Table 4-1 .

**Table 4-1 . General Information Requests**

| Request | Purpose |
|---|---|
| **PEXGetAncillaryBufferInfo** | Returns information about the ancillary buffers of the given visuals and pixmap depths. |
| **PEXGetExtensionInfo** | Gives PEX version, identity, and encoding information. |
| **PEXGetEnumeratedTypeInfo** | Lists the enumerated types available. |
| **PEXGetImpDepConstants** | Returns values of server-dependent constants. |
| **PEXMatchRendererTargets** | Identifies PEX-capable drawables. |
| **PEXQueryColorApproximation** | Verifies a color approximation. |

### 4.1.1 Get Ancillary Buffer Information

**Name**

**PEXGetAncillaryBufferInfo**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *drawable_id*: | DRAWABLE_ID |
| *num_visuals*: | CARD16 |
| *num_depths*: | CARD16 |
| *visuals*: | LISTofVISUALID |
| *depths*: | LISTofCARD8 |

**Reply**

| | |
|---|---|
| *visual_info*: | LISTofANCILLARY_BUFFER_INFO |
| *depth_info*: | LISTofANCILARY_BUFFER_INFO |

**Errors**

Drawable
*drawable_id* does not identify a valid drawable.

FloatingPointFormat
The specified floating-point format is not supported.

Value
A visual ID does not identify a visual for the screen.

**Description**

Z-, alpha, and accumulation buffer information is returned for the requested visuals given by *visuals* and/or pixmap depths given by *depths.* The screen is indicated by the drawable identified by *drawable_id.*

If an ancillary buffer is not supported for a particular visual or pixmap depth, the buffer's depth is returned as 0, and the contents of the corresponding minimum and maximum value or component depth fields are undefined.

**Encoding: PEXGetAncillaryBufferInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 133 | PEX opcode |
| 2 | 3+*m*+(*n*+*p*)/4 | request length |
| 4 | DRAWABLE_ID | drawable ID |
| 2 | CARD16 | number of visuals (*m*) |
| 2 | CARD16 | number of pixmap depths (*n*) |
| 4**m* | LISTofVISUALID | visual IDs |
| *n* | LISTofCARD8 | pixmap depths |
| *p* | | *p*=pad(*n*) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *a**(*m*+*n*) | reply length |
| 24 | | unused |
| *a**(*m*+*n*) | LISTofANCILLARY_BUFFER_INFO | buffer information (see ANCILLARY_BUFFER_INFO defined below) |

#### 4.1.1.1 Type ANCILLARY_BUFFER_INFO

ANCILLARY_BUFFER_INFO : [Z_depth, Z_format, alpha_depth, alpha_format : CARD8,
Z_min, Z_max, alpha_min, alpha_max : VALUE,
accum_depth : CARD8,
accum_r_depth, accum_g_depth, accum_b_depth, accum_alpha_depth : CARD8]

**Encoding: ANCILLARY_BUFFER_INFO**

| # of Bytes†1 | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | Z-buffer depth |
| 1 | CARD8 | | Z-buffer format |
| | 0 | *IntegerBuffer* | |
| | 1 | *FloatBuffer* | |
| 1 | CARD8 | | alpha buffer depth |
| 1 | CARD8 | | alpha buffer format |
| | 0 | *IntegerBuffer* | |
| | 1 | *FloatBuffer* | |
| 4 | VALUE†2 | | Z-buffer minimum value |
| 4 | VALUE†2 | | Z-buffer maximum value |
| 4 | VALUE†3 | | alpha buffer minimum value |
| 4 | VALUE†3 | | alpha buffer maximum value |
| 1 | CARD8 | | accumulation buffer depth |
| 1 | CARD8 | | accumulation buffer red component depth |
| 1 | CARD8 | | accumulation buffer green component depth |
| 1 | CARD8 | | accumulation buffer blue component depth |
| 1 | CARD8 | | accumulation buffer alpha component depth |
| 3 | | | unused |

†1) Size *a* of an ANCILLARY_BUFFER_INFO is determined as follows. The base size is 12. The size increases by 8 if the *Z_format* field is *IntegerBuffer* or by 2**fp* if the *Z_format* field is *FloatBuffer*. The size similarly increases according to the *alpha_format* field.
†2) The data type, either CARD32 or FLOAT, corresponding to VALUE is indicated by field *Z_format*.
†3) The data type, either CARD32 or FLOAT, corresponding to VALUE is indicated by the field *alpha_format*.

### 4.1.2 Get Extension Information

5.0

**Name**

**PEXGetExtensionInfo**

**Request**

| | |
|---|---|
| *client_protocol_major_version*: | CARD16 |
| *client_protocol_minor_version*: | CARD16 |

**Reply**

| | |
|---|---|
| *protocol_major_version*: | CARD16 |
| *protocol_minor_version*: | CARD16 |
| *vendor*: | LISTofCARD8 |
| *release_number*: | CARD32 |
| *subset_info*: | CARD32 |

**Errors**

None

**Description**

This request returns information describing the server's PEX extension.

The *client_protocol_major_version* and the *client_protocol_minor_version* indicate what version of the protocol the client expects the server to implement. The protocol version numbers returned indicate the protocol the PEX extension actually supports. This might not equal the version sent by the client. A PEX extension can (but need not) support more than one version simultaneously. The *protocol_major_version* and the *protocol_minor_version* are a mechanism to support future revisions of the PEX protocol which may be necessary. In general, the major version would increment for incompatible changes, and the minor version would increment for small, upward-compatible changes. Servers that support the protocol defined in this document will return a *protocol_major_version* of five, and a *protocol_minor_version* of two.

A string of ISO-LATIN1 characters identifying the vendor of the PEX server extension is returned in *vendor*. The characters in the string are implementation-dependent. An implementation-dependent number further identifying the product is returned in *release_number*.

*subset_info* indicates the PEX subsets supported by the server. The top 16 bits of *subset_info* are implementation-dependent. If the 16 low-order bits are all 0 then the server supports all PEX requests. If the lowest-order bit is 1, then the server supports the Immediate Rendering subset, listed in §4.1.2.1. If the next to lowest bit is 1, the server supports only the

PHIGS Workstation requests, listed in §4.1.2.3. If the third lowest bit is 1, the server supports the PEX structure requests, listed in §4.1.2.2. A Request error is generated when a server receives a request that it does not support.

**Encoding: PEXGetExtensionInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 1 | PEX opcode |
| 2 | 2 | request length |
| 2 | CARD16 | client protocol major version |
| 2 | CARD16 | client protocol minor version |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | ($n$+$p$)/4 | reply length |
| 2 | CARD16 | protocol major version |
| 2 | CARD16 | protocol minor version |
| 4 | CARD32 | release number |
| 4 | CARD32 | length of vendor name (bytes) ($n$) |
| 4 | CARD32 | subset info |
| | 0x00000000 *complete* 5.0 | |
| | 0x00000001 *immediate_rendering* 5.0 | |
| | 0x00000002 *phigs_wks_only* 5.0 | |
| | 0x00000004 *structure_rendering* 5.1 | |
| | 0x00000005 (imm and struct rendering) 5.1 | |
| 8 | | unused |
| $n$ | LISTofCARD8 | vendor |
| $p$ | | $p$=pad($n$) |

### 4.1.2.1 Immediate Rendering Subset

The immediate rendering subset consists of:

- All the General requests (those defined in this chapter).
- All Lookup Table requests (§7), Name Set requests (§8), Pipeline Context requests (§9), Font requests (§13), Texture Mapping (§14) and Accumulation Buffer requests (§15).
- All Renderer requests (§10) except **PEXRenderElements**, **PEXRenderNetwork**, and **PEXAccumulateState**.
- The Renderer Picking requests **PEXBeginPickOne**, **PEXEndPickOne**, **PEXBeginPickAll**, and **PEXEndPickAll** (§11).
- All output commands (§6).

#### 4.1.2.2 Structure Rendering Subset

The structure rendering subset consists of:

- All the General requests (those defined in this chapter).
- All Lookup Table requests (§ 7), Name Set requests (§ 8), Pipeline Context requests (§ 9), Font requests (§ 13), Texture Mapping (§ 14) and Accumulation Buffer requests (§ 15).
- All Structure (§ 12) and Search Context (§ 16) requests.
- All Renderer requests (§ 10) except **PEXRenderOutputCommands**.
- Requests **PEXPickOne** and **PEXPickAll** (§ 11).
- All output commands (§ 6).

#### 4.1.2.3 The PHIGS Workstation Only Subset

The PHIGS workstation subset consists of all version 5.1 or older requests and output commands described in the following chapters. Support for 5.2 and newer requests and output commands is undefined and implementation-dependent (see §17.1).

- All the General requests (those defined in this chapter).
- All Lookup Table requests (§ 7), Name Set requests (§ 8), and Font requests (§ 13).
- All Structure (§ 12) and Search Context (§ 16) requests.
- All PHIGS Workstation requests (§ 17).
- All Workstation Picking requests (§ 18).
- All output commands (§ 6).

### 4.1.3 Get Enumerated Type Information

**Name**

**PEXGetEnumeratedTypeInfo**

**Request**

| | |
|---|---|
| *drawable_id*: | DRAWABLE_ID |
| *enum_types*: | LISTofCARD16 |
| *item_mask*: | BITMASK |

**Reply**

| | |
|---|---|
| *types*: | LISTofENUM_DATA |

**Errors**

Drawable
*drawable_id* does not identify a valid drawable.

Match
The indicated drawable is not supported by the server’s PEX extension.

Value
One or more of the enumerated-type identifiers is invalid.

**Description**

The supported values of the PEX enumerated types indicated in *enum_types* are returned. These values are available for drawables with the same root window, depth, and visual ID as the drawable indicated by *drawable_id*. The *item_mask* indicates the information to return for each enumerated type specified, as shown in Table 4-2 .

**Table 4-2 . Information returned from PEXGetEnumeratedTypeInfo Constant Values**

| Value | Interpretation | Description |
|---|---|---|
| 0 | *(no values)* | The number of supported values for each enumerated type is returned. |
| 1 | *index* | The list of supported values for each enumerated type is returned. |
| 2 | *mnemonic* | The list of mnemonics of the supported values for each enumerated type is returned. Each mnemonic is a STRING in the ISO 8859/1 (Latin1) character set. The mnemonics are returned exactly as they are defined in the “Mnemonic” column of each table throughout this document defining an enumerated type, unless the mnemonic is implementation-dependent. In this case the mnemonic will be given as “(implementation-dependent)”. Implementation-dependent mnemonics may have the same value as another member of that enumerated class or may be an unregistered value. |
| 3 | *index and mnemonic* | The list of supported values and their mnemonics are returned for each enumerated type. |

The PEX enumerated types are listed in Table 4-3 . The value, name, and the location of the description of each enumerated type is given. The descriptions of each enumerated type are located throughout this document in tables titled “Enumerated Type *Foo*”. Each of these tables defines the registered enumerated type values and mnemonics, and gives a description of each value. Enumerated type values less than zero are implementation-dependent. Values greater than the registered values are reserved for future registration.

**Table 4-3 . Enumerated Types Constant Values**

| Value | Symbol | Location of Description |
|---|---|---|
| 1 | *MarkerType* | §6.9.58 5.0 |
| 2 | *ATextStyle* | §6.9.6 5.0 |
| 3 | *InteriorStyle* | §6.9.84 5.0 |
| 4 | *HatchStyle* | §6.9.85 5.0 |
| 5 | *LineType* | §6.9.48 5.0 |
| 6 | *SurfaceEdgeType* | §6.9.82 5.0 |
| 7 | *PickDeviceType* | §11.1.2 5.0 |
| 8 | *PolylineInterpMethod* | §6.9.68 5.0 |
| 9 | *CurveApproxMethod* | §6.9.25 5.0 |
| 10 | *ReflectionModel* | §6.9.88 5.0 |
| 11 | *SurfaceInterpMethod* | §6.9.86 5.0 |
| 12 | *SurfaceApproxMethod* | §6.9.75 5.0 |
| 13 | *ModelClipOperator* | §6.9.60 5.0 |
| 14 | *LightType* | §7.2.10 5.0 |
| 15 | *ColorType* | §3.7.6 5.0 |
| 16 | *FloatFormat* | §3.7.5 5.0 |
| 17 | *HLHSRMode* | §10.2.20 5.0 |
| 18 | *PromptEchoType* | §18.2 5.0 |
| 19 | *DisplayUpdateMode* | §17.6.13 5.0 |
| 20 | *ColorApproxType* | §7.2.3 5.0 |
| 21 | *ColorApproxModel* | §7.2.3 5.0 |
| 22 | *GDP* | §6.7.16 5.0 |
| 23 | *GDP3* | §6.7.15 5.0 |
| 24 | *GSE* | §6.9.32 5.0 |
| 25 | *TrimCurveApproxMethod* | §6.7.24 5.0 |
| 26 | *RenderingColorModel* | §6.9.72 5.0 |
| 27 | *ParametricSurfaceCharacteristics* | §6.9.63 5.0 |
| 28 | *Escape* | §5.1 5.1 |
| 29 | *PickOneMethod* | §11.1.3.1 5.1 |
| 30 | *PickAllMethod* | §11.1.3.2 5.1 |
| 31 | *AlphaBlendingFunction* | §6.9.2 5.2 |
| 32 | *DataMappingMethod* | §6.9.27 5.2 |
| 33 | *HighlightMethod* | §7.2.8 5.2 |

**Table 4-3 . Enumerated Types Constant Values (Continued)**

| Value | Symbol | Location of Description |
|---|---|---|
| 34 | *HLHSRIdentifier* | §6.9.38 |
| 35 | *LineCapStyle* | §6.9.43 |
| 36 | *LineJoinStyle* | §6.9.46 |
| 37 | *MarkerGlyphType* | §7.2.13 |
| 38 | *PrimAntiAliasBlendFunction* | §6.9.69 |
| 39 | *TransparencyMethod* | §10.2.40 |
| 40 | *ZTest* | §6.9.101 |
| 41 | *EchoMethod* | §10.2.13 |
| 42 | *TMPerspectiveCorrection* | §6.9.96 |
| 43 | *TMResourceHints* | §6.9.97 |
| 44 | *TMSampleFrequency* | §6.9.98 |
| 45 | *TMRenderingControl* | §7.2.17.1 |
| 46 | *TMParameterizationMethod* | §7.2.17.2.1 |
| 47 | *TMCompositionMethod* | §7.2.17.3 |
| 48 | *TMSampleMethod* | §7.2.17.4.1 |
| 49 | *TMBoundaryCondition* | §7.2.17.4.2 |
| 50 | *TMBoundaryClampMethod* | §7.2.17.4.3 |
| 51 | *TMDimension* | §14.1.1 |
| 52 | *TMTexelType* | §14.1.1 |
| 53 | *SupportedTexelResources* | §14.1.2 |

**Encoding: PEXGetEnumeratedTypeInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 2 | PEX opcode |
| 2 | 4+(2**n*+*p*)/4 | request length |
| 4 | DRAWABLE_ID | drawable identifier |
| 4 | BITMASK | item mask |
| | 0x00000000 (counts) | |
| | 0x00000001 *index* | |
| | 0x00000002 *mnemonic* | |
| | 0x00000003 (index and mnemonic) | |
| 4 | CARD32 | number of requested enumerated types (*n*) |
| 2**n* | LISTofCARD16 | requested enumerated types (See Table 4-3 for the values.) |
| *p* | | *p*=pad(2**n*) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *m*/4 | reply length |
| 4 | CARD32 | number of lists (*n*) |
| 20 | | unused |
| *m* | LISTofENUM_DATA†1 | enumerated types values |

†1) The contents of ENUM_DATA depend on *item_mask*. If *item_mask* = 0, then only the counts are returned and ENUM_DATA is a CARD32. If *item_mask* = 1, then only the indices are returned and ENUM_DATA is a LISTofINT16. If *item_mask* = 2, then only the mnemonics are returned and ENUM_DATA is a LISTofSTRING_ALIGNED_PACKED. If *item_mask* = 3, then the indices and mnemonics are returned and ENUM_DATA is a LISTofENUM_DESC (see below).

### 4.1.3.1 Type ENUM_DESC

ENUM_DESC : [value : ENUM_TYPE_INDEX,
mnemonic : STRING]

**Encoding: ENUM_DESC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | ENUM_TYPE_INDEX | enumerated type value |
| 2+*n*+*p* | STRING_UNALIGNED_PACKED | enumerated type mnemonic |

Note the enumerated type value is a 16-bit field. In the case of the *Escape*, *GDP3*, *GDP*, and *GSE* enumerated type classes, the members are identifying a 32-bit identifier. The descriptions of those classes specify the mapping between the 16-bit enumerated type value and the 32-bit identifier. Since many 32-bit identifiers can map to the same 16-bit enumerated type value, the client must also consider the corresponding enumerated type mnemonics to uniquely identify the members of these classes.

### 4.1.4 Get Implementation-dependent Constants

**Name**

**PEXGetImpDepConstants**

**Request**

*fp_format*: FLOAT_FORMAT
*drawable_example*: DRAWABLE_ID
*names*: LISTofCONSTANT_NAME

**Reply**

*constants*: LISTofVALUE

**Errors**

Drawable
*drawable_example* does not identify a valid drawable.

FloatingPointFormat
The specified floating-point format is not supported.

Match
The indicated drawable is not supported by the server’s PEX extension.

Value
One or more of the values in *names* is unrecognized.

**Description**

The implementation-dependent constants dependent on a particular PEX server extension and applying to drawables with the same root, depth, and visual ID as *drawable_example* are returned. Table 4-4 lists the implementation-dependent constants defined by PEX. Implementations may define additional constants.

The constants are returned in the same order in *constants* as specified in *names*. There is one return value in *constants* for each requested value in *names*. Floating-point values are returned in the format specified by *fp_format*.

Implementation-defined constants have a CONSTANT_NAME with the high-order bit set (0x8000). The standard convention is for the registered vendor ID to be in the next seven bits: (0x8000 | (vendor << 8) | *imp_dep_id*).

**Table 4-4 . Implementation-Dependent Constants Constant Values**

| Value | Symbol | Type | Description |
|---|---|---|---|
| 1 | *DitheringSupported* | CARD32 | *True* if the dithering is supported for color approximation; *False* if dithering is not supported.<br>The value returned is 1 if *DitheringSupported* is *True*, 0 if *DitheringSupported* is *False*. |
| 2 | *MaxEdgeWidth* | CARD32 | Width (in pixels) of thickest edge that can be drawn. |
| 3 | *MaxLineWidth* | CARD32 | Width (in pixels) of thickest line or curve that can be drawn. |

**Table 4-4 . Implementation-Dependent Constants Constant Values**

| Value | Symbol | Type | Description |
|---|---|---|---|
| 4 | *MaxMarkerSize* | CARD32 | Largest dimension (either height or width, in pixels) of largest marker that may be drawn. (This maximum is exclusive of the marker type *Dot* which is always drawn as the smallest displayable point). 5.0 |
| 5 | *MaxModelClipPlanes* | CARD32 | Maximum number of modeling clip planes that are used simultaneously during model clipping. 5.0 |
| 6 | *MaxNamesetNames* | CARD32 | Maximum number of names allowed in a name set. The possible names are those between zero and (*MaxNamesetNames* - 1), inclusive. 5.0 |
| 7 | *MaxNonAmbientLights* | CARD32 | Maximum number of simultaneous non-ambient light sources that are considered in reflectance calculations. 5.0 |
| 8 | *MaxNURBOrder* | CARD32 | Maximum spline order supported for B-spline curves and surfaces. 5.0 |
| 9 | *MaxTrimCurveOrder* | CARD32 | Maximum spline order supported for B-spline trimming curves. 5.0 |
| 10 | *MinEdgeWidth* | CARD32 | Width (in pixels) of thinnest edge that can be drawn. 5.0 |
| 11 | *MinLineWidth* | CARD32 | Width (in pixels) of thinnest line or curve that can be drawn. 5.0 |
| 12 | *MinMarkerSize* | CARD32 | Largest dimension (either height or width, in pixels) of smallest marker that may be drawn. (This minimum is exclusive of the marker type *Dot* which is always drawn as the smallest displayable point). 5.0 |
| 13 | *NominalEdgeWidth* | CARD32 | Width (in pixels) of “standard” edge. 5.0 |
| 14 | *NominalLineWidth* | CARD32 | Width (in pixels) of “standard” line or curve. 5.0 |
| 15 | *NominalMarkerSize* | CARD32 | Largest dimension (either height or width, in pixels) of “standard” marker. 5.0 |
| 16 | *NumSupportedEdgeWidths* | CARD32 | Number of supported edge widths.<br>A value of 0 indicates that all edge widths between and including *MinEdgeWidth* and *MaxEdgeWidth*, including fractional widths, are supported. 5.0 |
| 17 | *NumSupportedLineWidths* | CARD32 | Number of supported line widths.<br>A value of 0 indicates that all line widths between and including *MinLineWidth* and *MaxLineWidth*, including fractional widths, are supported. 5.0 |
| 18 | *NumSupportedMarkerSizes* | CARD32 | Number of supported marker sizes.<br>A value of 0 indicates that all marker sizes between and including *MinMarkerSize* and *MaxMarkerSize*, including fractional sizes, are supported. 5.0 |

**Table 4-4 . Implementation-Dependent Constants Constant Values**

| Value | Symbol | Type | Description |
|---|---|---|---|
| 19 | *BestColorApproxValues* | CARD32 | *ColorApproxPowersOf2* indicates that color approximation multipliers that are powers of two are likely to cause better performance than values that are not powers of 2. *ColorApproxAnyValues* indicates that there is no such advantage for power-of-2 multipliers.<br>The value returned is 0 if *BestColorApproxValues* is *ColorApproxAnyValues*, 1 if *BestColorApproxValues* is *ColorApproxPowersOf2*. 5.0 |
| 20 | *TransparencySupported* | CARD32 | *True* if the transmission coefficient is utilized in the reflection calculation. *False* otherwise.<br>The value returned is 1 if *TransparencySupported* is *True*, 0 if *TransparencySupported* is *False*. 5.0 |
| 21 | *DoubleBufferingSupported* | CARD32 | *True* if PEX workstation resources can provide double buffering.<br>The value returned is 1 if *DoubleBufferingSupported* is *True*, 0 if *DoubleBufferingSupported* is *False*. 5.0 |
| 22 | *ChromaticityRedU* | FLOAT | CIEYUV *u* chromaticity coefficient for the red channel of the (properly adjusted) display device. 5.0 |
| 23 | *ChromaticityRedV* | FLOAT | CIEYUV *v* chromaticity coefficient for the red channel of the (properly adjusted) display device. 5.0 |
| 24 | *LuminanceRed* | FLOAT | CIEYUV luminance value for the red channel of the (properly adjusted) display device. 5.0 |
| 25 | *ChromaticityGreenU* | FLOAT | CIEYUV *u* chromaticity coefficient for the green channel of the (properly adjusted) display device. 5.0 |
| 26 | *ChromaticityGreenV* | FLOAT | CIEYUV *v* chromaticity coefficient for the green channel of the (properly adjusted) display device. 5.0 |
| 27 | *LuminanceGreen* | FLOAT | CIEYUV luminance value for the green channel of the (properly adjusted) display device. 5.0 |
| 28 | *ChromaticityBlueU* | FLOAT | CIEYUV *u* chromaticity coefficient for the blue channel of the (properly adjusted) display device. 5.0 |
| 29 | *ChromaticityBlueV* | FLOAT | CIEYUV *v* chromaticity coefficient for the blue channel of the (properly adjusted) display device. 5.0 |
| 30 | *LuminanceBlue* | FLOAT | CIEYUV luminance value for the blue channel of the (properly adjusted) display device. 5.0 |
| 31 | *ChromaticityWhiteU* | FLOAT | CIEYUV *u* chromaticity coefficient for the reference white of the (properly adjusted) display device. 5.0 |
| 32 | *ChromaticityWhiteV* | FLOAT | CIEYUV *v* chromaticity coefficient for the reference white of the (properly adjusted) display device. 5.0 |
| 33 | *LuminanceWhite* | FLOAT | CIEYUV luminance value for the reference white of the (properly adjusted) display device. 5.0 |

**Table 4-4 . Implementation-Dependent Constants Constant Values**

| Value | Symbol | Type | Description |
|---|---|---|---|
| 34 | *MaxHitsEventSupported* | CARD32 | *True* if the *MaxHitsReached* event is generated when the maximum number of pick hits is exceeded during renderer immediate mode pick all operations.<br>The value returned is 1 if *MaxHitsEventSupported* is *True*, 0 if *MaxHitsEventSupported* is *False*. 5.1 |
| 35 | *ConcentrationClampSupported* | CARD32 | *True* if the concentration clamping region is applied in the reflection calculation involving a light of type *WCS_Spot. False* otherwise.<br>The value returned is 1 if *ConcentrationClampSupported* is *True*, 0 if *ConcentrationClampSupported* is *False*. 5.2 |
| 36 | *DitheringControllable* | CARD32 | *True* if *DitheringSupported* is *True* and dithering can be turned on and off with the dither field of a color approximation table. *False* if *DitheringSupported* is *False*, or if *DitheringSupported* is *True* and dithering cannot be disabled.<br>The value returned is 1 if *DitheringControllable* is *True*, 0 if *DitheringControllable* is *False*. 5.2 |
| 37 | *EyepointUsed* | CARD32 | *True* if the eyepoint defined in the view table is used. *False* if the server ignores the eyepoint defined in the view table and calculates an implicit eyepoint itself.<br>The value returned is 1 if *EyepointUsed* is *True*, 0 if *EyepointUsed* is *False*. 5.2 |
| 38 | *MaxMarkerVertices* | CARD32 | Maximum number of vertices which can be used to define a marker of marker glyph type *Stroke* in the marker glyph table. 5.2 |
| 39 | *MaxNumPickResults* | CARD32 | Maximum number of pick path elements that can be accumulated. 5.2 |
| 40 | *PrimAntiAliasSupported* | CARD32 | The 4-byte value returned is really of type PRIM_AA_BITMASK (§6.9.69). Each defined bit corresponding to a type of supported primitive antialiasing is set to 1. The defined bits set to 0 correspond to types of primitive antialiasing which are not supported. All undefined bits are set to 0. 5.2 |
| 41 | *QuadraticAttenSupported* | CARD32 | *True* if the quadratic attenuation factor is utilized in the reflection calculation involving a light of type *WCS_Point* or *WCS_Spot. False* otherwise.<br>The value returned is 1 if *QuadraticAttenSupported* is *True*, 0 if *QuadraticAttenSupported* is *False*. 5.2 |
| 42 | *PixelColorSupported* | CARD32 | *True* if the renderer's *pixel color mode* attribute behaves as specified when its value is *On. False* if the *pixel color mode* has no effect.<br>The value returned is 1 if *PixelColorSupported* is *True*, 0 if *PixelColorSupported* is *False*. 5.2 |

**Table 4-4 . Implementation-Dependent Constants Constant Values**

| Value | Symbol | Type | Description |
|---|---|---|---|
| 43 | *MaxTextureMaps* | CARD32 | Maximum number of texture maps in the pipeline state’s *active textures* or *bf active textures* lists which may be active for a single facet of a primitive. |
| 44 | *MaxFastTMSize* | CARD32 | Maximum size in any direction of the base level of a texture map, which allows optimized implementation. Larger texture maps may be defined, but will be treated in an implementation-dependent manner (e.g. they may be scaled down to fit into the maximum size, or they may used as defined with a non-optimized rendering path). A value of zero indicates all sizes are equally optimized. |
| 45 | *PowerOfTwoTMRequired* | CARD32 | *True* if the sizes in all directions of all texel arrays defining a texture map must be powers of two. Otherwise, *False* is returned.<br>The value returned is 1 if *PowerOfTwoTMRequired* is *True*, 0 if *PowerOfTwoTMRequired* is *False*. |
| 46 | *SquareTMRequired* | CARD32 | *True* if each level of a texture map must be equally sized in all directions. Otherwise, *False* is returned.<br>The value returned is 1 if *SquareTMRequired* is *True*, 0 if *SquareTMRequired* is *False*. |
| 47 | *TMSamplingHintsUsed* | CARD32 | *True* if the sampling hints field of the texture mapping lookup table is used to make adjustments when sampling a texture map. Otherwise, *False* is returned.<br>The value is 1 if *TMSamplingHintsUsed* is *True*, 0 if *TMSamplingHintsUsed* is *False*. |

**Encoding: PEXGetImpDepConstants**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 3 | PEX opcode |
| 2 | 4+(2*$n$+$p$)/4 | request length |
| 2 | INT16 | floating-point format |
| 2 | | unused |
| 4 | DRAWABLE_ID | example drawable identifier |
| 4 | CARD32 | number of imp-dep constant names ($n$) |
| 2*$n$ | LISTofCONSTANT_NAME | imp-dep constant names (see below for CONSTANT_NAME) |
| $p$ | | $p$=pad(2*$n$) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $m$ | reply length |
| 24 | | unused |
| 4*$m$ | LISTofVALUE[†1] | imp-dep constants values (length implicit) |

†1) The data type, either CARD32 or FLOAT, corresponding to VALUE is defined with each implementation-dependent constant.

CONSTANT_NAME : CARD16 (See Table 4-4 for the registered values.)

### 4.1.5 Match Renderer Targets 5.1

**Name**

**PEXMatchRendererTargets**

**Request**

| | |
|---|---|
| *drawable_id*: | DRAWABLE_ID |
| *depth*: | CARD8 |
| *target_type*: | MATCH_DRAW_TYPE |
| *visual*: | VISUALID |
| *max_targets*: | CARD32 |

**Reply**

| | |
|---|---|
| *targets*: | LISTofRENDERER_TARGET |

**Errors**

Drawable
*drawable_id* does not identify a valid drawable.

Value
1) *visual* does not identify a visual for the screen.
2) *type* is not one of the defined types.

**Description**

This request returns the visual, depth, and drawable-type combinations that the server supports on the screen of the drawable identified by *drawable_id*. Only those renderer targets that match the specified values of *depth*, *target_type*, and *visual* are returned. A value of *DontCare* (0) for any of these three parameters indicates that renderer targets for all supported values of that parameter are to be returned. (A value of *DontCare* for all three thus causes all renderer targets available to be returned.) If *max_targets* is non-zero, its value gives the maximum number of targets to return. If this is less than the number of matching targets, then it is implementation dependent which of the matching targets are returned. If *max_targets* is 0, all matching targets are returned.

**Encoding: PEXMatchRendererTargets**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 94 | PEX opcode |
| 2 | 5 | request length |
| 4 | DRAWABLE_ID | drawable ID |
| 1 | CARD8 | depth |
| 1 | | unused |
| 2 | MATCH_DRAW_TYPE | target type (MATCH_DRAW_TYPE defined below) |
| 4 | VISUALID | visual ID |
| 4 | CARD32 | max targets |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 2**n* | reply length |
| 24 | | unused |
| 8**n* | LISTofRENDERER_TARGET | targets (length implicit) (RENDERER_TARGET defined below) |

MATCH_DRAW_TYPE : {*DontCare*, *Window*, *Pixmap*, *Buffer*[†1]}

**Encoding: MATCH_DRAW_TYPE**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | drawable type |
| | 0 *DontCare* 5.1 | |
| | 1 *Window* 5.1 | |
| | 2 *Pixmap* 5.1 | |
| | 3 *Buffer* 5.1 | |

†1) DBE Buffer

RENDERER_TARGET : [depth : CARD8,
type : MATCH_DRAW_TYPE,
visual : VISUALID]

**Encoding: RENDERER_TARGET**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | depth |
| 1 | | unused |
| 2 | MATCH_DRAW_TYPE | type |
| 4 | VISUALID | visual ID |

### 4.1.6 Query Color Approximation

**Name**

**PEXQueryColorApproximation**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *drawable_example*: | DRAWABLE_ID |
| *capx*: | COLOR_APPROX_TABLE_ENTRY (§7.2.3) |

**Reply**

| | |
|---|---|
| *capx_supported*: | BOOLEAN |
| *list_is_exhaustive*: | BOOLEAN |
| *supported_capxs*: | LISTofCOLOR_APPROX_TABLE_ENTRY (§7.2.3) |

**Errors**

FloatingPointFormat
The specified floating-point format is not supported.

Drawable
The specified drawable ID is not valid.

Match
The specified drawable is not supported.

Value
1) The color model does not name a valid color approximation color model.
2) The specified dither value is not a member of {On, Off}.

**Description**

This request indicates whether the specified color approximation given in *capx* is supported by drawables with the same visual ID (if the drawable has one), root, and depth as *drawable_example*. If the color approximation is not supported, the reply also provides a list of alternative color approximations that are supported and indicates whether that list is exhaustive.

If the color approximation is supported, *capx_supported* is returned as *True*, no list of alternative color approximations is returned, and the returned value of *list_is_exhaustive* is undefined. If the capx is not supported, *capx_supported* is returned as *False* and at least one supported color approximation is returned in *supported_capxs*.

Floating-point values are returned in the format specified by *fp_format*. The boolean *list_is_exhaustive* is returned as *True* if the returned list of supported color approximations contains all color approximations supported by drawables with the same visual ID (if the drawable has one), root, and depth as *drawable_example*.

**Encoding: PEXQueryColorApproximation**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 105 | PEX opcode |
| 2 | 10+3*$fp$/4 | request length |
| 2 | FLOAT_FORMAT | floating point format |
| 2 | | unused |
| 4 | DRAWABLE_ID | example drawable id |
| 28+3*$fp$ | COLOR_APPROX_TABLE_ENTRY (§7.2.3) | color approximation |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | (3*$fp$/4 + 7)*$n$ | reply length |
| 1 | BOOLEAN | given color approximation is supported |
| 1 | BOOLEAN | list of supported approximations is exhaustive |
| 2 | | unused |
| 4 | CARD32 | number of returned color approximations ($n$) |
| 16 | | unused |
| (3*$fp$+28)* $n$ | LISTofCOLOR_APPROX_TABLE_ENTRY (§7.2.3) | list of supported color approximations |

# 5 Escapes

## 5.1 General

The escape requests provide a standard interface to functionality not defined by PEX but available in a particular PEX server extension. There are two escape requests (see Table 5-1 ), one with a reply and one without a reply. Escapes that return information use the escape with reply.

**Table 5-1 . Escape Requests**

| Request | Purpose |
|---|---|
| **PEXEscape** | Invoke an escape that does not have an associated reply. |
| **PEXEscapeWithReply** | Invoke an escape that returns a reply. |

Each escape request holds a 32-bit unsigned integer ID identifying the escape to invoke, and a data record containing any data needed by that escape. The entity defining the escape defines the ID and the encoding of the escape's data record. The encoding is defined such that the total length of the escape request is a multiple of four bytes.

A Value error is generated if an escape not available in the PEX implementation is requested, if the incorrect escape request was used to invoke the escape (the **PEXEscape** request was used for an escape that has a reply), or if an escape is not supported for the particular drawable involved in the request. Only errors and events defined by PEX or core X are potentially generated by an escape. To provide functionality that requires new errors or events, an implementation must define a new X extension.

### 5.1.1 Escape IDs

The escape ID is chosen by the entity (person, vendor, organization) defining the escape, subject to the convention defined below. Escapes may be defined by the PEX specification. An *escape_id* has bit 0x80000000 zeroed if the escape is defined by PEX. Otherwise bit 0x80000000 is set to indicate the escape is a vendor extension. The vendor-specific identifier is encoded in bits 0x7FFF0000, and the escape identifier in bits 0x0000FFFF. The **PEXGetEnumeratedTypeInfo** request is used to query class *Escape*, the escapes supported by the server. However **PEXGetEnumeratedTypeInfo** can only return 16-bit enumerated type values, so it encodes an escape value with bit 0x8000 zeroed if the escape is defined by PEX or set if the escape is a vendor extension, the vendor-specific identifier is taken from bits 0x007F0000 and encoded in bits 0x7F00, and the identifier is taken from bits 0x000000FF and encoded in bits 0x00FF. Where differing 32-bit escape identifiers map to the same 16-bit *Escape* enumerated type value, the enumerated type mnemonics distinguish them. Vendor IDs are assigned by the X Consortium and maintained in the Consortium's registry.

### 5.1.2 Enumerated Type *Escape*

Enumerated type *Escape,* inquired with request **PEXGetEnumeratedTypeInfo**, lists the registered and vendor-specific escapes supported. The registered escapes are shown in Table 5-2 .

**PEXGetEnumeratedTypeInfo** can return only 16-bit enumerated type values. It encodes an escape ID with the high-order bit (0x8000) indicating whether the escape is registered or vendor-specific, the vendor-specific identifier (if applicable) in the next lower order 7 bits (0x7F00), and the lowest-order 8 bits (0x00FF) contain the vendor-selected value identifying the escape. If differing escapes map to the same enumerated type value, their enumerated type mnemonics distinguish them.

**Table 5-2 . Enumerated Type *Escape***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *SetEchoColor* | This escape sets the renderer’s *echo color* attribute. When the renderer’s *echo mode* is *Echo*, the renderer’s *echo method* determines whether the *echo color* is used to render primitives.<br>It is strongly recommended the renderer’s *echo color* be modified with the usual requests used to modify renderer attributes rather than with this escape. The escape is provided to provide compatibility with PEX versions prior to 5.2.<br>A *Renderer* error is issued if the renderer identifier given is invalid. A *FloatingPointFormat* error is issued if an unsupported floating point format is given. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. 5.1 |

*Escape* value *SetEchoColor* data :

[fp_format : FLOAT_FORMAT,
rdr_id : RENDERER_ID,
color : COLOR_SPECIFIER]

**Encoding:** ***Escape*** Value *SetEchoColor* Data Record

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer resource identifier |
| 4+*sc* | COLOR_SPECIFIER | echo color (see COLOR for *sc*) |

### 5.1.3 Escape

5.1

**Name**

**PEXEscape**

**Request**

| | |
|---|---|
| *escape_id*: | CARD32 |
| *escape_data*: | data |

**Errors**

Value
The escape indicated by *escape_id* is not supported, has a reply associated with it, or one or more of the values in *escape_data* is invalid.

*Escape-specific*
X11 and PEX errors appropriate to the escape. The cause of such errors are described in the escape's definition.

**Description**

This request invokes the escape identified by *escape_id*. Only escapes that do not have a reply associated with them may be invoked with this request. The content and encoding of *escape_data* are specific to each escape.

**Encoding: PEXEscape**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 95 | PEX opcode |
| 2 | 2+*n*†1 | request length |
| 4 | CARD32 | escape ID |
| 4**n* | | escape data (length implicit) |

†1) *n* may be 0.

### 5.1.4 Escape With Reply

5.1

**Name**

**PEXEscapeWithReply**

**Request**

| | |
|---|---|
| *escape_id*: | CARD32 |
| *escape_data*: | data |

**Reply**

| | |
|---|---|
| *escape_id*: | CARD32 |
| *escape_reply_data*: | data |

**Errors**

Value

The escape indicated by *escape_id* is not supported, does not have a reply associated with it, or one or more of the values in *escape_data* is invalid.

*Escape-specific*

X11 and PEX errors appropriate to the escape. The cause of such errors are described in the escape’s definition.

**Description**

This request invokes the escape identified by *escape_id*. Only escapes that have a reply associated with them may be invoked with this request. The content and encoding of *escape_data* and *escape_reply_data* are specific to each escape. For all escapes, *escape_reply_data* is defined such that the entire reply length is a multiple of 4 bytes.

**Encoding: PEXEscapeWithReply**

| # of Bytes | Data Type or Defined Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 96 | PEX opcode |
| 2 | $2+n^{\dagger 1}$ | request length |
| 4 | CARD32 | escape ID |
| $4*n$ | | escape data |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $m^{\dagger 2}$ | reply length |
| 4 | CARD32 | escape ID |
| 20 | | escape reply data |
| $m$ | | escape reply data |

†1) $n$ may be 0.
†2) $m$ must be a multiple of 4. It may be 0.

# 6 Output Commands

## 6.1 General

Output commands define the output primitives that PEX displays, and the attribute values to apply to those primitives. Each output command type specifies a different form of output primitive or an attribute to control the appearance or rendering of output primitives. Everything displayed by PEX is first defined with an output command. Output commands are passed to the server with the requests **PEXRenderOutputCommands** and **PEXStoreElements**. This section specifies the output commands that define output primitives.

Output commands are processed immediately by a renderer or stored in a structure as structure elements. When sent to the server within a **PEXRenderOutputCommands** request, they are executed immediately. When sent within a **PEXStoreElements** request, they are stored in the structure indicated in that request. Output commands stored in a structure may be subsequently passed to a renderer and displayed using request **PEXRenderNetwork**, or through an *Execute Structure, Conditional Execute Structure, Include Structure,* or *Conditional Include Structure* output command within a **PEXRenderOutputCommands** request. Output commands stored in a structure may be retrieved with the **PEXFetchElements** request.

During rendering, unsupported attribute values are mapped to their default values, and out-of-range or undefined table indices are mapped to their fallback values. No errors are generated for these conditions. See Section 6.3 for the conditions under which *OutputCommand* errors are generated.

## 6.2 Output Command Format

All output commands have at least a four-byte header containing a 16-bit unsigned integer identifying the output command type and a 16-bit unsigned integer identifying the output command's total length, including the header. The length is the number of 4-byte words in the output command. Output commands whose data may be very long have encodings which explicitly contain a mechanism analogous to the Big Request Extension to X (BRE). For these OCs, if the 16-bit length field is insufficient, a zero is placed in the 16-bit length field and a 4-byte length field immediately follows, making the header an 8-byte header.

Data particular to the type of output command immediately follows the header.

### 6.2.1 Vendor-specific Output Command Types

The high-order bit (0x8000) of the 16-bit output command type, when set, indicates a vendor specific output command type. PEX vendors may define output command types not defined by PEX. If such an output command is not recognized by a server, the output command is not rendered and has no effect on the rendered image. It is, however, stored as a structure element if the output command is within a **PEXStoreElements** request. No floating-point conversion or byte-swapping is performed on vendor-specific output commands not supported by the server.

The ability to define vendor-specific output commands is maintained within PEX version 5.2 only for backwards compatibility with previous versions. PEX implementations should use the *Generalized Drawing Primitive (§6.7.15), Generalized Drawing Primitive 2D (§6.7.16),* and *Generalized Structure Element* (§6.9.32) output commands to define vendor-specific output command types.

### 6.2.2 Floating-point Format in Output Commands

Each request that sends or returns output commands has a parameter of type FLOAT_FORMAT. For requests that send output commands to the server, this parameter indicates the floating-point format of floating-point values in the request's output commands. For requests that cause a reply to be returned, the parameter also indicates the floating-point format the server is to use for floating-point values in output commands within the reply.

If a request specifies a floating point format that is not supported by the PEX server, a *FloatingPointFormat* error is reported and none of the output commands within that request are processed by a renderer or stored in a structure, even if the output commands do not contain floating point values.

### 6.2.3 Normal Vectors

Several output commands have normal vector parameters. Normal vectors in these output commands should be unit vectors. The effect of rendering output primitives with non-unit length normals is implementation dependent. See also §2.1.1.1.1.

### 6.2.4 Output Command Specification Syntax

When embedded in descriptions, output command names are capitalized with separating spaces (e.g., Fill Area Set). Output command parameters are lower case, use the underscore (_) for separation, and are printed in italics (e.g., *color_type*).

Output commands are described as follows:

**Name**

| | |
|---|---|
| *arg1_name*: | data type |
| *arg2_name*: | data type |

The functional description of the output command.

(Primitives only) The attribute sets (§6.5) and clipping rules (§6.6) which apply to the primitive.

(Attributes only) The attribute set (§6.5) to which the attribute belongs.

The situations under which an *OutputCommand* error arises.

The protocol encoding.

### 6.2.5 “With Data” Output Primitives Extensibility

Those primitives whose name includes the “with Data” clause share a mechanism for the optional inclusion of data. One or more bitmasks are used to indicate what optional data is present. The bitmask values therefore determine the format of the primitive’s encoding.

In version 5.2 an *fp_data* bit is added to many of these bitmasks, specifying whether a list of floating-point data values is optionally included. A required field, ignored if the corresponding *fp_data* bit is not set, is also added to these primitives to indicate the length of the floating-point data lists.

The floating-point data lists provide some extensibility to the data associated with these primitives. The intention is for new functionality requiring additional primitive data to make use of the floating-point data lists if at all possible. The new functionality then references into the floating-point data lists to obtain the primitive data values required. For example, texture mapping parameterization method *FloatData* uses an index into a primitive’s vertex floating-point data lists, and the dimension of the texture map, to obtain texture map coordinates.

As new functionality requiring new floating-point primitive data is designed, a method for indexing into the appropriate floating-point data lists should be included. The avoid run-time errors, the behavior is implementation-dependent if the floating-point data is in some way inadequate, e.g., the number of values is insufficient or a value is out of range.

## 6.3 Output Command Errors

The following error conditions within an output command cause an *OutputCommand* error to be generated for the request containing the offending output command:

a) A parameter expected to contain a value from a fixed enumeration (e.g., TEXT_PATH) contains an undefined value.

b) A bit that PEX defines to be *unused* within a bitmask is not zero.

c) A color type not supported by the PEX server is requested.

d) The length of an output command, when added to that of previous information and output commands within a request, would exceed the given request length of that request.

Output commands containing such errors are neither rendered nor stored in a structure. When the erroneous output command is within a list of output commands, all output commands prior to the erroneous one in the list are processed normally (rendered or stored in a structure), but the offending output command and all output commands subsequent to it in the list are ignored.

## 6.4 Output Command Descriptions

Sections 6.7, 6.9 and 6.10 give the output command definitions, descriptions, and encodings for output primitive output commands. The descriptions indicate the effect of the output command when the command is processed by a renderer—when the renderer receives it directly from the client or when it is read from a structure that is being rendered or executed. Output commands are not interpreted by a renderer at the time they are stored in a structure, and have no effect at that time other than to change the contents of the structure.

## 6.5 Output Command Primitives and their Attribute Sets

Each output primitive has one or more sets of attributes applied to it. The tables in this section define the attributes sets, listing the attribute output command (OC) and renderer attribute (RA) members of each attribute set. The output primitives affected by each attribute set are also listed. The definition of each output primitive, attribute output command, and renderer attribute refers to the attribute sets defined in this section.

Bundled attributes are those controlled by aspect source flags (ASFs). The ASF determines whether the value used for the attribute is taken from a pipeline state field (modified with an OC) or from a bundle table entry. In the tables below, beneath an OC which modifies the index into a bundle table, bundled attributes' OCs are indented and marked with a (*b*). See §2.1.4.1 for further discussion on bundled attributes.

**Table 6-1 . Annotation Attributes**

| **Attributes** | **Primitive OCs Affected** |
|---|---|
| Annotation Text Alignment OC | Annotation Text |
| Annotation Text Height OC | Annotation Text 2D |
| Annotation Text Path OC | |
| Annotation Text Style OC | |
| Annotation Text Up Vector OC | |

**Table 6-2 . Annotation and Text Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Text Bundle Index OC | Annotation Text |
| Character Expansion OC (*b*) | Annotation Text 2D |
| Character Spacing OC (*b*) | Text |
| Text Color OC (*b*) | Text 2D |
| Text Color Index OC (*b*) | |
| Text Font Index OC (*b*) | |
| Text Precision OC (*b*) | |
| Text Bundle Table RA | |
| Text Font Table RA | |

**Table 6-3 . Curve Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Line Bundle Index OC | Arc Area Set with Data |
| Curve Approximation Method OC (*b*) | Arc Set with Data |
| Line Bundle Table RA | Cone Set with Data |
| | Non-uniform B-spline Curve |
| | Non-uniform B-spline Curve with Color |
| | Sphere Set with Data |

**Table 6-4 . Curved Surface Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Interior Bundle Index OC | Non-uniform B-spline Surface |
| Parametric Surface Bundle Index OC†1 | Non-uniform B-spline Surface with Data |
| Surface Approximation Method OC (*b*) | |
| Interior Bundle Table RA | |
| PHIGS Interior Bundle Mode RA | |

†1) The renderer’s *PHIGS interior bundle mode* determines whether the pipeline state’s *interior bundle index* or *parametric surface bundle index* selects the indicated members of the interior bundle table.

**Table 6-5 . Data Mapping Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Data Mapping Index OC†1 | Arc Area Set with Data |
| Data Mapping Method OC (*b*) | Cone Set with Data |
| Backface Data Mapping Index OC | Fill Area Set with Data (V5.0) |
| Backface Data Mapping Method OC (*b*) | Fill Area Set with Data (V5.2) |
| Data Mapping Table RA | Independent Triangles with Data |

**Table 6-5 . Data Mapping Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| | Indexed Independent Triangles with Data |
| | Quadrilateral Mesh with Data (V5.0) |
| | Quadrilateral Mesh with Data (V5.2) |
| | Set of Fill Area Sets with Data (V5.0) |
| | Set of Fill Area Sets with Data (V5.2) |
| | Sphere Set with Data |
| | Triangle Fan with Data |
| | Triangle Strip with Data (V5.0) |
| | Triangle Strip with Data (V5.2) |
| | Triangle Strip Fan Set with Data |

†1) Both the data mapping index and backface data mapping index select from the data mapping bundle table.

**Table 6-6 . Edge Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Edge Bundle Table | Arc Area Set with Data |
| Surface Edge Bundle Index OC | Fill Area |
| Surface Edge Color OC (*b*) | Fill Area 2D |
| Surface Edge Color Index OC (*b*) | Fill Area Set |
| Surface Edge Flag OC (*b*) | Fill Area Set 2D |
| Surface Edge Type OC (*b*) | Fill Area with Data |
| Surface Edge Width OC (*b*) | Fill Area Set with Data (V5.0) |
| | Fill Area Set with Data (V5.2) |
| | Independent Triangles with Data |
| | Indexed Independent Triangles with Data |
| | Non-uniform B-spline Surface |
| | Non-uniform B-spline Surface with Data |
| | Quadrilateral Mesh with Data (V5.0) |
| | Quadrilateral Mesh with Data (V5.2) |
| | Set of Fill Area Sets with Data (V5.0) |
| | Set of Fill Area Sets with Data (V5.2) |
| | Triangle Fan with Data |
| | Triangle Strip with Data (V5.0) |
| | Triangle Strip with Data (V5.2) |
| | Triangle Strip Fan Set with Data |

**Table 6-7 . General Surface Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Facet Culling Mode OC | Arc Area Set with Data |
| Facet Distinguish Flag OC | Cell Array |
| Interior Bundle Index OC | Cell Array 2D |
| Backface Reflectance Bundle Index OC†1 | Cone Set with Data |
| Backface Surface Reflection Attributes OC (*b*) | Extended Cell Array |
| Backface Surface Reflection Model OC (*b*) | Fill Area |
| Reflectance Bundle Index OC†2 | Fill Area 2D |
| Surface Reflection Attributes OC (*b*) | Fill Area Set |
| Surface Reflection Model OC (*b*) | Fill Area Set 2D |
| Interior Bundle Table RA | Fill Area with Data |
| Light Source State OC | Fill Area Set with Data (V5.0) |
| Light Table RA | Fill Area Set with Data (V5.2) |
| PHIGS Interior Bundle Mode RA | Independent Triangles with Data |
| Vertex Order OC | Indexed Independent Triangles with Data |
| | Non-uniform B-spline Surface |
| | Non-uniform B-spline Surface with Data |
| | Quadrilateral Mesh with Data (V5.0) |
| | Quadrilateral Mesh with Data (V5.2) |
| | Set of Fill Area Sets with Data (V5.0) |
| | Set of Fill Area Sets with Data (V5.2) |
| | Sphere Set with Data |
| | Triangle Fan with Data |
| | Triangle Strip with Data (V5.0) |
| | Triangle Strip with Data (V5.2) |
| | Triangle Strip Fan Set with Data |

†1) The renderer’s *PHIGS interior bundle mode* determines whether the pipeline state’s *interior bundle index* or *reflectance bundle index* selects the indicated members of the interior bundle table.
†2) The renderer’s *PHIGS interior bundle mode* determines whether the pipeline state’s *interior bundle index* or *backface reflectance bundle index* selects the indicated members of the interior bundle table.

**Table 6-8 . Global Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Add Names to Name Set OC | (all primitives) |
| Alpha Blending Function OC | |
| Background Color RA | |
| Clear Alpha RA | |
| Clear Image RA | |
| Clear Z-buffer RA | |
| Clip List RA | |

**Table 6-8 . Global Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Color Approximation Index OC | |
| Color Approximation Table RA | |
| Color Table RA | |
| Depth Cue Index OC | |
| Depth Cue Table RA | |
| Drawing Function OC | |
| Echo Mode RA | |
| Global Transform OC | |
| Global Transform with Hints OC | |
| Global Transform 2D OC | |
| Global Transform 2D with Hints OC | |
| Highlight Exclusion Name Set RA | |
| Highlight Inclusion Name Set RA | |
| Highlight Index OC | |
| Highlight Table RA | |
| HLHSR Identifier OC | |
| HLHSR Mode RA | |
| Individual ASF OC | |
| Invisibility Exclusion Name Set RA | |
| Invisibility Inclusion Name Set RA | |
| Local Transform OC | |
| Local Transform with Hints OC | |
| Local Transform 2D OC | |
| Local Transform 2D with Hints OC | |
| Model Clip Indicator OC | |
| Model Clip Volume OC | |
| Model Clip Volume 2D OC | |
| NPC Subvolume RA | |
| Pick Identifier OC | |
| Pipeline Context RA | |
| Pixel Color Mode RA | |
| Plane Mask OC | |
| Primitive Antialiasing OC | |
| Remove Names from Name Set OC | |
| Renderer State RA | |
| Rendering Color Model OC | |
| Restore Model Clipping Volume OC | |
| Transparency Method RA | |
| Transparency Thresholds RA | |

**Table 6-8 . Global Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| View Index OC | |
| View Table RA | |
| Viewport RA | |
| Z-Buffer Test OC | |

**Table 6-9 . Line Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Line Bundle Index OC | Arc Set with Data |
| Line Cap Style OC (*b*) | Non-uniform B-spline Curve |
| Line Color OC (*b*) | Non-uniform B-spline Curve with Color |
| Line Color Index OC (*b*) | Polyline |
| Line Join Style OC (*b*) | Polyline 2D |
| Line Type OC (*b*) | Polyline Set with Data (V5.0) |
| Line Width OC (*b*) | Polyline Set with Data (V5.2) |
| Line Bundle Table RA | |
| Line Miter Limit OC | |

**Table 6-10 . Marker Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Marker Bundle Index OC | Marker |
| Marker Color OC (*b*) | Marker 2D |
| Marker Color Index OC (*b*) | Marker with Data |
| Marker Scale Factor OC (*b*) | |
| Marker Type OC (*b*) | |
| Marker Bundle Table RA | |

**Table 6-11 . Parametric Surface Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Interior Bundle Index OC | Non-uniform B-spline Surface |
| Parametric Surface Bundle Index OC[†1] | Non-uniform B-spline Surface with Data |
| Parametric Surface Characteristics OC (*b*) | |
| Interior Bundle Table RA | |
| PHIGS Interior Bundle Mode RA | |

†1) The renderer's *PHIGS interior bundle mode* determines whether the pipeline state's *interior bundle index* or *parametric surface bundle index* selects the indicated members of the interior bundle table.

**Table 6-12 . Shaded Line Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Line Bundle Index OC | Arc Set with Data |
| Polyline Interpolation Method OC (*b*) | Polyline |
| Line Bundle Table RA | Polyline 2D |
| | Polyline Set with Data (V5.0) |
| | Polyline Set with Data (V5.2) |

**Table 6-13 . Styled and Shaded Surface Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Interior Bundle Index OC | Arc Area Set with Data |
| Surface Color OC (*b*) | Cone Set with Data |
| Surface Color Index OC (*b*) | Fill Area |
| Surface Interior Style OC (*b*) | Fill Area 2D |
| Surface Interior Style Index OC (*b*) | Fill Area Set |
| Surface Interpolation Method OC (*b*) | Fill Area Set 2D |
| Backface Interior Bundle Index OC†1 | Fill Area with Data |
| Backface Surface Color OC (*b*) | Fill Area Set with Data (V5.0) |
| Backface Surface Color Index OC (*b*) | Fill Area Set with Data (V5.2) |
| Backface Surface Interior Style OC (*b*) | Independent Triangles with Data |
| Backface Surface Interior Style Index OC (*b*) | Indexed Independent Triangles with Data |
| Backface Surface Interpolation Method OC (*b*) | Non-uniform B-spline Surface |
| Interior Bundle Table RA | Non-uniform B-spline Surface with Data |
| Backface Surface Active Textures OC | Quadrilateral Mesh with Data (V5.0) |
| Pattern Reference Point OC | Quadrilateral Mesh with Data (V5.2) |
| Pattern Reference Point and Vectors OC | Set of Fill Area Sets with Data (V5.0) |
| Pattern Size OC | Set of Fill Area Sets with Data (V5.2) |
| Pattern Table RA | Sphere Set with Data |
| PHIGS Interior Bundle Mode RA | Triangle Fan with Data |
| Surface Active Textures OC | Triangle Strip with Data (V5.0) |
| Texture Mapping Perspective Correction OC | Triangle Strip with Data (V5.2) |
| Texture Mapping Resource Hints OC | Triangle Strip Fan Set with Data |
| Texture Mapping Sample Frequency OC | |
| Texture Mapping Table RA | |

†1) The renderer’s *PHIGS interior bundle mode* determines whether the pipeline state’s *interior bundle index* or *backface interior bundle index* selects the indicated members of the interior bundle table.

**Table 6-14 . Text Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Character Height OC | Text |
| Character Up Vector OC | Text 2D |
| Text Alignment OC | |
| Text Path OC | |

**Table 6-15 . Vendor-Specific Attributes**

| Attributes | Primitive OCs Affected |
|---|---|
| Generalized Structure Element OC | (vendor-specific) |

## 6.6 Clipping Rules

### 6.6.1 Annotation Text Clipping

Annotation text primitives are not displayed if their origin is outside the currently defined clipping volume. If the origin is not clipped, the string is clipped according to the text clipping rules and the origin-offset connection line enabled by the pipeline state's *annotation text style* is clipped according to the line clipping rules. Model clipping is considered when it is determined whether the origin is clipped, but model clipping is not applied to the string or to the connection line.

### 6.6.2 Line Clipping

Line primitives are not displayed if they are outside the currently defined clipping volume. Lines which cross the clipping volume are clipped and only the portions inside the clipping volume are displayed.

### 6.6.3 Marker Clipping

Markers are not displayed if they are outside the currently defined clipping volume. The clipping of a marker whose position is inside the clipping volume but whose rendering is outside the clipping volume is implementation-dependent.

### 6.6.4 Surface Clipping

Surface primitives are not displayed if they are outside the currently defined clipping volume. Surfaces which cross the clipping volume limits are clipped; only the portions inside the clipping volume are displayed. A boundary created by the surface crossing the clipping volume is considered an interior boundary and therefore is not edged if edging is enabled. However, a boundary caused by clipping is rendered as part of the interior when the interior style is *Hollow*.

### 6.6.5 Text Clipping

Text is clipped depending on the pipeline state's *text precision*. If the precision is *String*, clipping is done in an implementation-dependent manner. If the precision is *Char*, clipping is done on at least a character-by-character basis. If the text precision is *Stroke*, clipping is performed at the clipping boundaries for each character.

## 6.7 Primitive Output Commands

This section lists the output commands that define output primitives.

### 6.7.1 Annotation Text

5.0

| | |
|---|---|
| *origin*: | COORD_3D |
| *offset*: | COORD_3D |
| *string*: | LISTofMONO_ENCODING |

This command causes an annotation text string to be rendered. An annotation text string is a set of characters which are rendered in an X-Y plane in normalized projection coordinates (NPCs). This guarantees annotation text is always parallel to the display surface. *origin* is transformed to NPCs where *offset* is then added to it to define the origin of the text local coordinate system. The X and Y axes of the text local coordinate system are parallel to and in the same direction as the X and Y axes of normalized projection coordinate space. Note the text local coordinate system has the same scale as the normalized projection coordinate space.

Usage of *string* is as defined for the Text primitive (§6.7.35).

The Annotation and Annotation and Text attribute sets and annotation text clipping are applied to Annotation Text.

**Encoding: Annotation Text**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 83 | output command number |
| 2 | $2+s+(6*fp+n)/4$ OR 0 | output command length |
| $k^{\dagger 1}$ | $3+s+(6*fp+n)/4$ | 32-bit output command length |
| *fp* | FLOAT | origin x (MCs) |
| *fp* | FLOAT | origin y (MCs) |
| *fp* | FLOAT | origin z (MCs) |
| *fp* | FLOAT | offset x (NPCs) |
| *fp* | FLOAT | offset y (NPCs) |
| *fp* | FLOAT | offset z (NPCs) |
| 2 | CARD16 | number of mono-encodings (*s*) |
| 2 | | unused |
| $4*s+n^{\dagger 2}$ | LISTofMONO_ENCODING | string |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) $n$ is the sum of the sizes of the mon-encodings. See MONO_ENCODING (§3.7.7.84) for the mono-encoding size.

### 6.7.2 Annotation Text 2D

5.0

*origin*: COORD_2D
*offset*: COORD_2D
*string*: LISTofMONO_ENCODING

This command causes an annotation text string to be rendered. The command is a 2D form of the Annotation Text output command (§6.7.1), and functions exactly as the 3D annotation text primitive except *origin* and *offset* are 2D values with the Z-coordinate implicitly zero.

**Encoding: Annotation Text 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 84 | output command number |
| 2 | 2+*s*+(4**fp*+*n*)/4 OR 0 | output command length |
| *k*†1 | 3+*s*+(4**fp*+*n*)/4 | 32-bit output command length |
| *fp* | FLOAT | origin x |
| *fp* | FLOAT | origin y |
| *fp* | FLOAT | offset x |
| *fp* | FLOAT | offset y |
| 2 | CARD16 | number of mono-encodings (*s*) |
| 2 | | unused |
| 4**s*+*n*†2 | LISTofMONO_ENCODING | string |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *n* is the sum of the sizes of the mon-encodings. See MONO_ENCODING (§3.7.7.84) for the mono-encoding size.

### 6.7.3 Arc Area Set with Data

*arc_area_type*: CARD16
*color_type*: COLOR_TYPE
*vert_attributes*: PRIM_DATA_ATTRIBUTES
*fp_data_count*: FP_DATA_COUNTS
*arc_areas*: LISTofVERTEX

This command causes each arc area in a set of arc areas to be rendered independently. The arc area types are listed in Table 6-16 .

**Table 6-16 . Arc Area Type Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 1 | *Circle* | A curve, which is everywhere equidistant from the center vertex in the $Z = 0$ plane, and its interior.<br>Note if a 3D circle is defined, the *Z* values are ignored.<br>For *Circle*, the radius field of each vertex's OPT_DATA must be defined.<br>If the radius of the circle is less than or equal to 0, the circle degrades to a dot at the center vertex. |
| 3 | *CircularClosedChordArc* | A portion of a circle defined by the radius, and a start angle (α) and a stop angle (β), in the $Z = 0$ plane, as follows:<br>$M = \langle radius, 0, 0 \rangle$ ,<br>$m = \langle 0, radius, 0 \rangle$ ,<br>$\sum_{A = \alpha}^{\beta} cosA \cdot M + sinA \cdot m$<br>If $\lvert\beta - \alpha\rvert > 2\pi$ , then $\beta = \alpha + 2\pi$ .<br>If $\beta < \alpha$ , then $\beta = \beta + 2\pi$ .<br>The arcs endpoints are joined by a segment.<br>Note if a 3D circular closed arc is defined, the *Z* values are ignored.<br>For *CircularClosedChordArc*, the radius and angles fields of each vertex's OPT_DATA must be defined.<br>If the radius of the circular closed arc is less than or equal to 0, the circular closed arc degrades to a dot at the center vertex. |
| 4 | *CircularClosedPieArc* | Same as *CircularClosedChordArc* except instead of the endpoints being joined by a segment, each endpoint of the arc is connected by a segment to the center vertex. |

**Table 6-16 . Arc Area Type Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 5 | *Ellipse* | A curve, defined by a center vertex and major and minor axes, and its interior. The major and minor axes define a parallelogram that circumscribes the ellipse and is anchored at the center vertex of the ellipse. Two sides of the parallelogram are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to the midpoints of the sides of the parallelogram. The ellipse is tangent to the sides of the parallelogram at their midpoints.<br>For *Ellipse*, the axes field of each vertex’s OPT_DATA must be defined.<br>If the length of both of the axes is equal to zero, the ellipse degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, or if the axes have non-zero lengths and are parallel, the ellipse degrades in an implementation-dependent way. |
| 7 | *EllipticalClosedChordArc* | A portion of an ellipse defined by a start angle ($\alpha$) and a stop angle ($\beta$) as follows:<br>$\sum_{A=\alpha}^{\beta} cosA \cdot MajorAxis + sinA \cdot MinorAxis$<br>If $\lvert\beta-\alpha\rvert > 2\pi$, then $\beta = \alpha + 2\pi$.<br>If $\beta < \alpha$, then $\beta = \beta + 2\pi$.<br>The arcs endpoints are joined by a segment.<br>For *EllipticalClosedChordArc*, the axes and angles fields of each vertex’s OPT_DATA must be defined.<br>If the length of both of the axes is equal to zero, the elliptical closed arc degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, or if the axes have non-zero lengths and are parallel, the elliptical closed arc degrades in an implementation-dependent way. |
| 8 | *EllipticalClosedPieArc* | Same as *EllipticalClosedChordArc* except instead of the endpoints being joined by a segment, each endpoint of the arc is connected by a segment to the center vertex. |

*vert_attributes* indicates the optional vertex data specified. The components of the *vert_attributes* bitmask are:

*color*: COLOR

*2D_coords*: (Indicates whether the Z-coordinate is specified)

*fp_data*: LISTofFLOAT

If any of these bits is 1, the corresponding data is indicated in *arc_areas*.

If vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored. The vertex color defines the color of the arc area centered at the vertex.

If the *2D_coords* bit is set, the arc area center vertex and (if applicable) major and minor axes have no Z component. Instead the Z components are assumed to be 0.0.

If the *fp_data* bit is set, then extended data is specified with each vertex. The *vertex_fp_count* in *fp_data_count* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If the *fp_data* bit is not set, *fp_data_count* is ignored.

The General Surface, Styled and Shaded Surface, Edge, Curve, and Data Mapping attribute sets, and surface clipping, are applied to Arc Area Set with Data.

An *OutputCommand* error is issued if the arc area type is not one of the registered values.

**Encoding: Arc Area Set with Data**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | 105 | | output command number |
| 2 | 5+$n$*$a$/4 OR 0 | | output command length |
| $k^{\dagger 1}$ | 6+$n$*$a$/4 | | 32-bit output command length |
| 2 | CARD16 | | arc area type |
| | 1 | *Circle* | |
| | 3 | *CircularClosedChordArc* | |
| | 4 | *CircularClosedPieArc* | |
| | 5 | *Ellipse* | |
| | 7 | *EllipticalClosedChordArc* | |
| | 8 | *EllipticalClosedPieArc* | |
| 2 | COLOR_TYPE | | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | | vertex attributes |
| 2 | | | unused |
| 4 | CARD32 | | number of arc areas ($n$) |
| 4 | FP_DATA_COUNTS | | the number of floats in the vertex floating-point data list |
| $n$*$v^{\dagger 3}$ | LISTofVERTEX | | arc data (see VERTEX for $v$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *2D_coords*, and *fp_data* are supported by Arc Area Set with Data.
†3) As noted by the definition of OPT_DATA, a component of VERTEX, the radius must be defined when the arc type is *Circle*, the radius and angles must be defined if the arc type is *CircularClosedChordArc* or *CircularClosedPieArc*, the axes must be defined if the arc type is *Ellipse*, and the axes and angles must be defined if the arc type is *EllipticalClosedChordArc* or *EllipticalClosedPieArc*.

### 6.7.4 Arc Set with Data

*arc_type*: CARD16
*color_type*: COLOR_TYPE
*vert_attributes*: PRIM_DATA_ATTRIBUTES
*fp_data_count*: FP_DATA_COUNTS
*arcs*: LISTofVERTEX

This command causes each arc in a set of arcs to be rendered independently. The arc types are listed in Table 6-17 .

**Table 6-17 . Arc Type Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 1 | *Circle* | A curve which is everywhere equidistant from the center vertex in the $Z = 0$ plane.<br>Note if a 3D circle is defined, the *Z* value are ignored.<br>If the radius of the circle is less than or equal to 0, the circle degrades to a dot at the center vertex. |
| 2 | *CircularArc* | A portion of a circle defined by the radius, and a start angle ($\alpha$) and a stop angle ($\beta$), in the $Z$= 0 plane, as follows:<br>$M = \langle radius, 0, 0 \rangle$ ,<br>$m = \langle 0, radius, 0 \rangle$ ,<br>$\sum_{A = \alpha}^{\beta} cosA \cdot M + sinA \cdot m$<br>If $\lvert\beta - \alpha\rvert > 2\pi$ , then $\beta = \alpha + 2\pi$ .<br>If $\beta < \alpha$ , then $\beta = \beta + 2\pi$ .<br>Note if a 3D circular arc is defined, the *Z* values are ignored.<br>If the radius of the circular arc is less than or equal to 0, the circular arc degrades to a dot at the center vertex. |
| 3 | *CircularClosedChordArc* | A circular arc in the $Z = 0$ plane whose endpoints are joined by a segment.<br>Note if a 3D circular closed arc is defined, the *Z* values are ignored.<br>If the radius of the circular closed arc is less than or equal to 0, the circular closed arc degrades to a dot at the center vertex. |
| 4 | *CircularClosedPieArc* | Same as *CircularClosedChordArc* except instead of the endpoints being joined by a segment, each endpoint of the arc is connected by a segment to the center vertex. |

**Table 6-17 . Arc Type Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 5 | *Ellipse* | A curve defined by a center vertex and major and minor axes. The major and minor axes define a parallelogram that circumscribes the ellipse and is anchored at the center vertex of the ellipse. Two sides of the parallelogram are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to the midpoints of the sides of the parallelogram. The ellipse is tangent to the sides of the parallelogram at their midpoints.<br>If the length of both of the axes is equal to zero, the ellipse degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, or if the axes have non-zero lengths and are parallel, the ellipse degrades in an implementation-dependent way. |
| 6 | *EllipticalArc* | A portion of an ellipse defined by a start angle ($\alpha$) and a stop angle ($\beta$) as follows:<br>$$\sum_{A=\alpha}^{\beta} cosA \cdot MajorAxis + sinA \cdot MinorAxis$$<br>If $\vert\beta - \alpha\vert > 2\pi$, then $\beta = \alpha + 2\pi$.<br>If $\beta < \alpha$, then $\beta = \beta + 2\pi$.<br>If the length of both of the axes is equal to zero, the elliptical arc degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, or if the axes have non-zero lengths and are parallel, the elliptical arc degrades in an implementation-dependent way. |
| 7 | *EllipticalClosedArc* | An elliptical arc whose endpoints are joined by a segment.<br>If the length of both of the axes is equal to zero, the elliptical closed arc degrades to a dot at the center vertex. If the length of just one of the axes is equal to zero, or if the axes have non-zero lengths and are parallel, the elliptical closed arc degrades in an implementation-dependent way. |
| 8 | *EllipticalClosedPieArc* | Same as *EllipticalClosedChordArc* except instead of the endpoints being joined by a segment, each endpoint of the arc is connected by a segment to the center vertex. |

*vert_attributes* indicates the optional vertex data specified. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *2D_coords*: | (Indicates whether the Z-coordinate is specified) |
| *fp_data*: | LISTofFLOAT |

If any of these bits is 1, the corresponding data is indicated in *arcs*.

If vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored. The vertex color defines the color of the arc centered at the vertex.

If the *2D_coords* bit is set, the arc center vertex and (if applicable) major and minor axes have no Z component. Instead the Z components are assumed to be 0.0.

If the *fp_data* bit is set, then extended data is specified with each vertex. The *vertex_fp_count* in *fp_data_count* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If the *fp_data* bit is not set, *fp_data_count* is ignored.

The Line, Shaded Line, and Curve attribute sets, and line clipping, are applied to Arc Set with Data.

An *OutputCommand* error is issued if the arc type is not one of the registered values.

**Encoding: Arc Set with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 106 | output command number |
| 2 | 5+$n$*$a$/4 OR 0 | output command length |
| $k$ †1 | 6+$n$*$a$/4 | 32-bit output command length |
| 2 | CARD16 | arc type |
| | 1 *Circle* | |
| | 2 *CircularArc* | |
| | 3 *CircularClosedChordArc* | |
| | 4 *CircularClosedPieArc* | |
| | 5 *Ellipse* | |
| | 6 *EllipticalArc* | |
| | 7 *EllipticalClosedChordArc* | |
| | 8 *EllipticalClosedPieArc* | |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of arcs ($n$) |
| 4 | FP_DATA_COUNTS | the number of floats in the vertex floating-point data list |
| $n$*$v$ †3 | LISTofVERTEX | arc data (see VERTEX for $v$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)

†2) Only bits *color*, *2D_coords*, and *fp_data* are supported by Arc Set with Data.

†3) As noted by the definition of OPT_DATA, a component of VERTEX, the radius must be defined when the arc type is *Circle*, the radius and angles must be defined if the arc type is *CircularArc*, *CircularClosedChordArc* or *CircularClosed-PieArc*, the axes must be defined if the arc type is *Ellipse*, and the axes and angles must be defined if the arc type is *EllipticalArc, EllipticalClosedChordArc* or *EllipticalClosedPieArc*.

### 6.7.5 Cell Array

5.0

| | |
|---|---|
| *point1*: | COORD_3D |
| *point2*: | COORD_3D |
| *point3*: | COORD_3D |
| *dx*: | CARD32 |
| *dy*: | CARD32 |
| *colors*: | LISTofTABLE_INDEX |

This command causes a cell array primitive to be rendered. A cell array is a parallelogram of equal-sized cells, each of which is a parallelogram and is rendered with a single color. The cell array parallelogram is defined by corner points *P, Q, R,* and ($Q_x+R_x-P_x$, $Q_y+R_y-P_y$, $Q_z+R_z-P_z$), where *P* is point1, *Q* is point2, *R* is point3, and *P*, *Q*, and *R* are defined in modeling coordinates. Implementations may minimally render just the transformed boundaries of the cell array parallelogram. The attributes applied to the boundary rendering are implementation-dependent.

There are *dx* by *dy* cells in the cell array. Each cell has width and height defined by:

$$width=\frac{\sqrt{(P_x-Q_x)^2+(P_y-Q_y)^2+(P_z-Q_z)^2}}{dx}$$ **Equation 6.1**

$$height=\frac{\sqrt{(P_x-R_x)^2+(P_y-R_y)^2+(P_z-R_z)^2}}{dy}$$ **Equation 6.2**

The colors are specified in a one-dimensional array of size $dx \times dy$. The colors are of type *Indexed*. Colors are stored in this array by rows, that is, the column number varies fastest as colors are stored into the array. The first color in the array is the color at the cell at the corner of *point1*, and subsequent colors represent the colors of cells proceeding to *point2*. If any color index is not defined in the color table, color index 1 is used. If color index 1 is not defined, the default value for the color table is used (§7.2.2).

The General Surface attribute set and surface clipping are applied to Cell Array.

An *OutputCommand* error is reported if *dx* or *dy* is zero.

**Encoding: Cell Array**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 99 | output command number |
| 2 | 3+(9*fp+2*q+p)/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 4+(9*fp+2*q+p)/4 | 32-bit output command length |
| 3*fp | COORD_3D | point1 |
| 3*fp | COORD_3D | point2 |
| 3*fp | COORD_3D | point3 |
| 4 | CARD32 | dx |
| 4 | CARD32 | dy |
| $2*q^{\dagger 2}$ | LISTofTABLE_INDEX | color indices |
| *p* | | *p*=pad(2*q) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) $q = dx \times dy$

### 6.7.6 Cell Array 2D

5.0

| | |
|---|---|
| *point1*: | COORD_2D |
| *point2*: | COORD_2D |
| *dx*: | CARD32 |
| *dy*: | CARD32 |
| *colors*: | LISTofTABLE_INDEX |

This command causes a cell array primitive to be rendered. The command is a 2D form of the Cell Array output command (§6.7.5), and differs from it only in the mechanism used to define the cell array parallelogram. Points *P* and *Q*, where *P* is *point1* and *Q* is *point2*, and *P* and *Q* are defined in modeling coordinates, define a rectangle which is taken to be aligned with the modeling coordinate axes. The Z-coordinate of *P* and *Q* is implicitly 0. Implementations may minimally render just the transformed boundaries of the cell array parallelogram. The attributes applied to the boundary rendering are implementation-dependent.

There are *dx* by *dy* cells in the cell array. Each cell has width and height defined by:

$$width=\frac{\sqrt{(P_x-Q_x)^2+(P_y-Q_y)^2}}{dx} \qquad \textbf{Equation 6.3}$$

$$height=\frac{\sqrt{(P_x-Q_x)^2+(P_y-Q_y)^2}}{dy} \qquad \textbf{Equation 6.4}$$

Usage of *colors* is as defined for the Cell Array primitive. Also see Cell Array for information, such as the attributes applied, which pertains to all cell array primitives.

An *OutputCommand* error is reported if *dx* or *dy* is zero.

**Encoding: Cell Array 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 100 | output command number |
| 2 | 3+*fp*+(2*s+p)/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 4+*fp*+(2*s+p)/4 | 32-bit output command length |
| 2**fp* | COORD_2D | point 1 |
| 2**fp* | COORD_2D | point 2 |
| 4 | CARD32 | dx |
| 4 | CARD32 | dy |
| $2*s^{\dagger 2}$ | LISTofTABLE_INDEX | color indices |
| *p* | | *p*=pad(2**s*) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) $s = dx \times dy$

### 6.7.7 Cone Set with Data

| | |
|---|---|
| *lower_radius*: | FLOAT |
| *upper_radius*: | FLOAT |
| *cap_lower*: | SWITCH |
| *cap_upper*: | SWITCH |
| *color_type*: | COLOR_TYPE |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_count*: | FP_DATA_COUNTS |
| *cones*: | LISTofVERTEX |

This command causes each cone in a set of cones to be rendered independently. Each cone is a three-dimensional surface whose upper and lower edges are defined by two circles which are perpendicular to the axis which passes through their centers. The sides of the cone are defined by a segment of the shortest possible length connecting the two circles as it is swept around their circumferences. The center vertex and radius of each circle are defined in modeling coordinates.

*cap_upper* controls whether the upper circle of the cones in the set is rendered as a surface with interior style *Solid*. Likewise *cap_lower* controls whether the lower circle of the cones in the set is rendered as a surface with interior style *Solid*. If a cap is *Off*, the bounding circle is treated as a surface with interior style *Hollow*.

*vert_attributes* indicates the optional data specified for the two center vertices of each cone. The components of the bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *radius*: | FLOAT |
| *fp_data*: | LISTofFLOAT |

If any of these bits is 1, the corresponding data is indicated.

If the *color* bit is set, a color is specified with each center vertex. The colors are of the type specified by *color_type*. The color associated with the center vertex of the lower circle is applied to the cone. The color associated with the center vertex of the upper circle is ignored. For the purposes of data mapping, each cone color is considered a “facet” color. If the *color* bit is not set, *color_type* is ignored.

If the *radius* bit is 1, the radii for each cone are defined in *cones* and parameters *lower_radius* and *upper_radius* are ignored. Otherwise parameters *lower_radius* and *upper_radius* define the common radii applied to all cones in the set, and radii are not defined with each cone. If a radius is less than zero or both radii are equal to zero, the results of rendering that cone are implementation-dependent.

If the *fp_data* bit is set, then extended data is specified with each center vertex. The *vertex_fp_count* in *fp_data_count* indicates the number of floating-point values in the LISTofFLOAT for each vertex. if the *fp_data* bit is not set, *fp_data_count* is ignored.

The General Surface, Styled and Shaded Surface, Curve, and Data Mapping attribute sets and surface clipping are applied to Cone Set with Data. However when interior style *Hollow* is applied to the cone set, nothing is rendered. The results when interior styles *Hatch* and *Pattern* are applied are implementation-dependent.

**Encoding: Cone Set with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 107 | output command number |
| 2 | 5+(2**fp* + *n**2**v*)/4 OR 0 | output command length |
| *k*†1 | 6+(2**fp* + *n**2**v*)/4 | 32-bit output command length |
| *fp* | FLOAT | lower radius |
| *fp* | FLOAT | upper radius |
| 2 | COLOR_TYPE | color type |
| 1 | SWITCH | cap lower |
| 1 | SWITCH | cap upper |
| 2 | PRIM_DATA_ATTRIBUTES†2 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of cones (*n*) |
| 4 | FP_DATA_COUNTS | the number of floats in the vertex floating-point data list |
| *n**2**v* | LISTofVERTEX | cone data (see VERTEX for *v*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *radius,* and *fp_data* are supported by Cone Set with Data.

### 6.7.8 Extended Cell Array

5.0

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *point1*: | COORD_3D |
| *point2*: | COORD_3D |
| *point3*: | COORD_3D |
| *dx*: | CARD32 |
| *dy*: | CARD32 |
| *colors*: | LISTofCOLOR |

This command causes a cell array primitive to be rendered. The command is an extended form of the Cell Array primitive (§6.7.5), and differs from it only in that *colors* may be indexed or direct color values, depending on *color_type*.

**Encoding: Extended Cell Array**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 101 | output command number |
| 2 | 4+9**fp*/4+*p***n* OR 0 | output command length |
| $k^{\dagger 1}$ | 5+9**fp*/4+*p***n* | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | | unused |
| 3**fp* | COORD_3D | point 1 |
| 3**fp* | COORD_3D | point 2 |
| 3**fp* | COORD_3D | point 3 |
| 4 | CARD32 | dx |
| 4 | CARD32 | dy |
| $p^{\dagger 2}*sc$ | LISTofCOLOR | colors (see COLOR for *sc*) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) $p = dx \times dy$

### 6.7.9 Fill Area

5.0

*shape*: SHAPE

*ignore_edges*: BOOLEAN

*vertices*: LISTofCOORD_3D

This command causes a fill area primitive to be rendered. A fill area is defined by a list of vertices in modeling coordinates which are joined together to form a planar surface. (The visual effect of a non-planar fill area is implementation-dependent.) The boundary of the fill area is defined by the segments joining vertex *i* to vertex *i+1*. A segment joins the last vertex to the first vertex to close the boundary. The boundary may be self-intersecting. The even-odd rule is used to determine the interior of the fill area.

*shape* is a hint indicating the complexity of the boundary. A value of *Unknown* indicates that nothing is known about the shape of the fill area, and the server may determine the hint itself. A value of *Complex* means that the boundary intersects itself. A value of *Nonconvex* means that the boundary does not self-intersect, but the fill area is not wholly convex. *Convex* indicates that all of the interior angles of the fill area are convex. Fill areas that are of a simpler complexity than indicated by *shape* are rendered correctly. Fill areas that are of a higher complexity than indicated by *shape* are drawn in an implementation-dependent manner.

*ignore_edges* specifies whether surface edge attributes are to be applied to the fill area. If *True*, no edges are drawn for the fill area. If *False*, edges are drawn using the current surface edge attributes if the pipeline state’s *surface edge flag* is *On*.

The General Surface, Styled and Shaded Surface, and Edge attribute sets and surface clipping are applied to Fill Area.

A Fill Area with fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

An *OutputCommand* error is reported if values in the *shape* or *ignore_edges* parameters are not recognized.

**Encoding: Fill Area**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 89 | output command number |
| 2 | 2+3**fp***p*/4 OR 0 | output command length |
| *k*†1 | 3+3**fp***p*/4 | 32-bit output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | | unused |
| 3**fp***p*†2 | LISTofCOORD_3D | vertices (length implicit) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *p* is the number of vertices in the fill area. It is derived from the output command length.

### 6.7.10 Fill Area 2D

5.0

| | |
|---|---|
| *shape*: | SHAPE |
| *ignore_edges*: | BOOLEAN |
| *vertices*: | LISTofCOORD_2D |

This command causes a fill area primitive to be rendered. The command is a 2D form of the Fill Area output command (§6.7.9), and differs from it only by containing 2D values for coordinates defined through *vertices*. The Z-coordinate of the coordinates is implicitly 0. Once these implicit values are associated with the primitive, it is rendered as a Fill Area primitive.

**Encoding: Fill Area 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 90 | output command number |
| 2 | 2+*fp***p*/2 OR 0 | output command length |
| *k*†1 | 3+*fp***p*/2 | 32-bit output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | | unused |
| 2**fp***p*†2 | LISTofCOORD_2D | vertices (length implicit) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *p* is the number of vertices in the fill area. It is derived from the output command length.

### 6.7.11 Fill Area Set

5.0

| | |
|---|---|
| *shape*: | SHAPE |
| *ignore_edges*: | BOOLEAN |
| *contour_hint*: | CONTOUR |
| *vertices*: | LISTofLISTofCOORD_3D |

This command causes a fill area set primitive to be rendered. A fill area set is defined by one or more lists of vertices in modeling coordinates. Each list forms one contour of the primitive. The even-odd rule applied to the combined contours determines the interior of the fill area set. The primitive is assumed by PEX to be planar. The visual effect of a non-planar fill area set is implementation-dependent.

*shape* is a hint indicating the complexity of the contours. A value of *Unknown* indicates that nothing is known about the shape of the contours, and the server may determine the hint self. A value of *Complex* means that one or more contours is self-intersecting. A value of *Nonconvex* means that no contour is self-intersecting, but some contour is not wholly convex. *Convex* indicates that all of the interior angles of all contours are convex. If the contours are simpler than indicated by *shape*, the primitive it still rendered correctly. The visual effect is implementation dependent if the contours have more complexity than indicated by *shape*.

*ignore_edges* specifies whether surface edge attributes are to be applied to the fill area. If *True*, no edges are drawn for the fill area. If *False*, edges are drawn using the current surface edge attributes if the pipeline state’s *surface edge flag* is *On*.

*contour_hint* provides information about the relationship of the contours. If *contour_hint* is *Disjoint*, all contours will be spatially disjoint; no overlapping or intersection occurs between any of the contours. If *contour_hint* is *Nested*, contours are either disjoint or wholly contained within another contour. If *contour_hint* is *Intersecting*, separate contours may have coincident or overlapping edges. If *contour_hint* is *Unknown*, nothing is known about the relationships between contours. The visual effect of an inaccurate value for *contour_hint* is implementation-dependent.

The General Surface, Styled and Shaded Surface, and Edge attribute sets and surface clipping are applied to Fill Area Set.

A fill area set primitive containing no contours or contours with fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

An *OutputCommand* error is reported if the *shape*, *ignore_edges*, or *contour_hint* parameter is invalid.

**Encoding: Fill Area Set**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 92 | output command number |
| 2 | 3+*n*+3**fp***m*/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 4+*n*+3**fp***m*/4 | 32-bit output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | CONTOUR | contour hint |
| 4 | CARD32 | number of contours (*n*) |
| 4**n*+3**fp** $m^{\dagger 2}$ | LISTofLISTofCOORD_3D | vertices |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *m* is the total number of vertices which define the fill area set. In the LISTofLISTofCOORD_3D, each fill area is defined by a CARD32 value indicating how many vertices it contains followed by the vertices (of type COORD_3D).

### 6.7.12 Fill Area Set 2D

5.0

| | |
|---|---|
| *shape*: | SHAPE |
| *ignore_edges*: | BOOLEAN |
| *contour_hint*: | CONTOUR |
| *vertices*: | LISTofLISTofCOORD_2D |

This command causes a fill area set primitive to be rendered. The command is a 2D form of the Fill Area Set output command (§6.7.11), and differs from it only by containing 2D values for coordinates defined through *vertices*. The Z coordinate of the coordinates is implicitly 0. Once these implicit values are associated with the primitive, it is rendered as an Fill Area Set primitive.

**Encoding: Fill Area Set 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 93 | output command number |
| 2 | $3+n+fp*m/2$ OR 0 | output command length |
| $k^{\dagger 1}$ | $4+n+fp*m/2$ | 32-bit output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | CONTOUR | contour hint |
| 4 | CARD32 | number of contours ($n$) |
| $4*n+2*fp*m^{\dagger 2}$ | LISTofLISTofCOORD_2D | vertices |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) $m$ is the total number of vertices which define the fill area set. In the LISTofLISTofCOORD_2D, each fill area is defined by a CARD32 value indicating how many vertices it contains followed by the vertices (of type COORD_2D).

### 6.7.13 Fill Area Set with Data (Version 5.0)

| | |
|---|---|
| *shape*: | SHAPE |
| *ignore_edges*: | BOOLEAN |
| *contour_hint*: | CONTOUR |
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *facet_data*: | OPT_DATA |
| *vertices*: | LISTofLISTofVERTEX |

This command causes a fill area set primitive to be rendered. The primitive is identical to the Fill Area Set primitive (§6.7.11) except this primitive allows the optional specification of facet colors and normals, and vertex colors, normals, and individual edge flags. The Fill Area Set with Data (V5.2) primitive (§6.7.14) is an expanded version of this primitive which additionally allows the optional specification of facet and vertex floating-point data lists and 2D vertices.

*facet_attributes* indicates the optional facet data specified. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |

If any of these bits is 1, the corresponding data is available in *facet_data*.

*vert_attributes* indicates the optional vertex data specified. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *edges*: | BITMASK |

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

*ignore_edges* specifies whether surface edge attributes are to be applied to the fill area. If *True*, no edges are drawn for the fill area. If *False*, edges are drawn according to the *edges* vertex attribute and the pipeline state's *surface edge flag. If the edges* bit of the vertex attributes is set, a surface edge flag is specified in the first bit (0x00000001) of the edges bitmask of each vertex. This edge data is used if the pipeline state's *surface edge flag* is *On*. For the *i*th vertex, the edge flag indicates whether to draw the edge between the *i*th and *i+1*th vertex. If the *edges* vertex attribute is not set, all the edges are drawn if the pipeline state's *surface edge flag* is *On*.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Fill Area Set with Data (V5.0).

A Fill Area Set with Data (V5.0) primitive containing no contours or contours with fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

Usage of *shape* and *contour_hint* is as defined for the Fill Area Set primitive (§6.7.11). An *OutputCommand* error is reported if the *shape*, *ignore_edges*, or *contour_hint* parameter is invalid.

**Encoding: Fill Area Set with Data (Version 5.0)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 94 | output command number |
| 2 | $5+n+(d+m*v)/4$ OR 0 | output command length |
| $k$†1 | $6+n+(d+m*v)/4$ | 32-bit output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | CONTOUR | contour hint |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of contours ($n$) |
| $d$ | OPT_DATA | facet data (see OPT_DATA for $d$) |
| $4*n+m*v$ | LISTofLISTofVERTEX | vertices (see VERTEX for $v$†4) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bits *color* and *normal* are supported by Fill Area Set with Data (V5.0).
†3) Only bits *color*, *normal*, and *edges* are supported by Fill Area Set with Data (V5.0).
†4) $m$ is the total number of vertices which define the fill area set. In the LISTofLISTofVERTEX, each fill area is defined by a CARD32 value indicating how many vertices it contains followed by the vertices and their data (of type VERTEX).

### 6.7.14 Fill Area Set with Data (Version 5.2) 5.2

| | |
|---|---|
| *shape*: | SHAPE |
| *ignore_edges*: | BOOLEAN |
| *contour_hint*: | CONTOUR |
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *facet_data*: | OPT_DATA |
| *vertices*: | LISTofLISTofVERTEX |

This command causes a fill area set primitive to be rendered. This primitive is identical to the Fill Area Set primitive (§6.7.11) except this primitive allows the optional specification of facet colors, normals, and floating-point data lists, and 2D or 3D vertices with a color, normal, individual edge flags, and a floating-point data list. The Fill Area Set with Data (V5.0) primitive (§6.7.13) is exactly like this primitive except it does not support facet and vertex floating-point data lists and 2D vertices.

*facet_attributes* indicates the optional facet data specified. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of these bits is 1, the corresponding data is available in *facet_data*.

*vert_attributes* indicates the optional vertex data specified. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *edges*: | BITMASK 5.0 |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) 5.2 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the facet attributes and/or vertex attributes is set, then extended data is specified with the facet and/or each vertex, respectively. If the *fp_data* facet attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for the facet. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. Each field in *fp_data_counts* is ignored if its corresponding *fp_data* bit is not set.

*ignore_edges* specifies whether surface edge attributes are to be applied to the fill area. If *True*, no edges are drawn for the fill area. If *False*, edges are drawn according to the *edges* vertex attribute and the pipeline state’s *surface edge flag. If the edges* bit of the vertex attributes is set, a surface edge flag is specified in the first bit (0x00000001) of the edges bitmask of each vertex. This edge data is used if the pipeline state’s *surface edge flag* is *On*. For the *i*th vertex, the edge flag indicates whether to draw the edge between the *i*th and *i+1*th vertex. If the *edges* vertex attribute is not set, all the edges are drawn if the pipeline state’s *surface edge flag* is *On*.

If the *2D_coords* vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Fill Area Set with Data (V5.2)

A Fill Area Set with Data (V5.2) primitive containing no contours or contours with fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

Usage of *shape* and *contour_hint* is as defined for the Fill Area Set primitive (§6.7.11). An *OutputCommand* error is reported if the *shape*, *ignore_edges*, or *contour_hint* parameter is invalid.

**Encoding: Fill Area Set with Data (Version 5.2)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 147 | output command number |
| 2 | $6+n+(d+m*v)/4$ OR 0 | output command length |
| $k^{\dagger 1}$ | $7+n+(d+m*v)/4$ | 32-bit output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | CONTOUR | contour hint |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of contours (*n*) |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex floating-point data lists |
| *d* | OPT_DATA | facet data (see OPT_DATA for *d*) |
| $4*n+m*v$ | LISTofLISTofVERTEX | vertices (see VERTEX for $v^{\dagger 4}$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Fill Area Set with Data (V5.2).
†3) Only bits *color*, *normal*, *edges*, *2D_coords*, and *fp_data* are supported by Fill Area Set with Data (V5.2).
†4) *m* is the total number of vertices which define the fill area set. In the LISTofLISTofVERTEX, each fill area is defined by a CARD32 value indicating how many vertices it contains followed by the vertices and their data (of type VERTEX).

### 6.7.15 Generalized Drawing Primitive

| | |
|---|---|
| *gdp_id*: | INT32 |
| *points*: | LISTofCOORD_3D |
| *data*: | data |

This command supports implementation-specific output primitives. Individual primitive types are identified by their unique *gdp_id*. For a particular type of GDP, the implementation must define the effect of rendering the primitive, the contents of *points* and *data* and what values will result in an *OutputCommand* error being issued, how the primitive is clipped, and which attributes affect it.

A *gdp_id* has its bit 32 set (0x80000000), the vendor-specific identifier encoded in bits 17 through 31 (0x7FFF0000), and the GDP identifier in bits 1 through 16 (0x0000FFFF). The **PEXGetEnumeratedTypeInfo** request is used to query class *GDP3*, the GDPs supported by the server. However **PEXGetEnumeratedTypeInfo** can only return 16-bit enumerated type values, so it encodes a *GDP3* value with bit 0x8000 set, the vendor-specific identifier is taken from bits 0x007F0000 of the 32-bit value and encoded in bits 0x7F00, and the GDP identifier is taken from bits 0x000000FF and encoded in bits 0x00FF. Where differing 32-bit GDP identifiers map to the same 16-bit *GDP3* enumerated type value, the enumerated type mnemonics distinguish them. The same is true for the *GDP* enumerated type class. Note *GDP3* and *GDP* are separate enumerations; there is not necessarily a correspondence between the GDP and GDP2D primitive types which share the same *gdp_id.*

Byte-swapping and floating-point conversion are only done for GDP types supported by the server.

If a particular GDP type is not supported by the server, it is stored in a structure when included in a **PEXStoreElements** request but has no visual effect when rendered. The unsupported GDP also has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Generalized Drawing Primitive**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 102 | output command number |
| 2 | $4+(3*fp*p+d+q)/4$ OR 0 | output command length |
| $k$[†1] | $5+(3*fp*p+d+q)/4$ | 32-bit output command length |
| 4 | INT32 | GDP identifier |
| 4 | CARD32 | number of points ($p$) |
| 4 | CARD32 | number of bytes of data ($d$) |
| $3*fp*p$ | LISTofCOORD_3D | points |
| $d$ | | data |
| $q$ | | $p$=pad($d$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)

### 6.7.16 Generalized Drawing Primitive 2D

| | |
|---|---|
| *gdp_id*: | INT32 |
| *points*: | LISTofCOORD_2D |
| *data*: | data |

This command supports implementation-specific 2D output primitives. Individual primitive types are identified by their unique *gdp_id*. For a particular type of GDP2D, the implementation must define the effect of rendering the primitive, the contents of *points* and *data* and what values will result in an *OutputCommand* error being issued, how the primitive is clipped, and which attributes affect it. The Z-coordinate of *points* is implicitly 0.

Usage of *gdp_id* is as defined for the Generalized Drawing Primitive (§6.7.15). The **PEXGetEnumeratedTypeInfo** request is used to query class *GDP*, the supported GDP2Ds. Note *GDP3* and *GDP* are separate enumerations; there is not necessarily a correspondence between the GDP and GDP2D primitive types which share the same *gdp_id*.

Byte-swapping and floating-point conversion are only done for GDP types supported by the server.

If a particular GDP type is not supported by the server, it is stored in a structure when included in a **PEXStoreElements** request but has no visual effect when rendered. The unsupported GDP also has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Generalized Drawing Primitive 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 103 | output command number |
| 2 | 4+(2**fp***p*+*d*+*q*)/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 5+(2**fp***p*+*d*+*q*)/4 | 32-bit output command length |
| 4 | INT32 | GDP ID |
| 4 | CARD32 | number of points (*p*) |
| 4 | CARD32 | number of bytes of data (*d*) |
| 2**fp***p* | LISTofCOORD_2D | points |
| *d* | | data |
| *q* | | *p*=pad(*d*) |

†1) *k* = ((output command length == 0) ? 4 : 0)

### 6.7.17 Independent Triangles with Data

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *num_verts*: | CARD32 |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *per_facet_data*: | LISTofOPT_DATA |
| *per_vert_data*: | LISTofVERTEX |

This command causes each triangle in a set of triangles to be rendered independently. A list of independent triangles is a collection of triangles defined a modeling coordinates vertex list. The *per_vert_data* is defined in a list of length *num_verts* such that the number of triangles rendered is floor(*num_verts* / 3)..

facet_attributes indicates the optional data specified for each triangle. The components of the facet_attributes bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *fp_data*: | LISTofFLOAT |

If any of the bits in *facet_attributes* is set indicating facet data is supplied, there are floor(*num_verts* / 3) entries of *per_facet_data* containing that facet data. Otherwise per_facet_data is empty.

vert_attributes indicates the optional data specified with each vertex. The components of the vert_attributes bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *edges*: | BITMASK |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) |
| *fp_data*: | LISTofFLOAT |

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by color_type. Otherwise color_type is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the facet and/or vertex attributes is set, then extended data is specified with each facet and/or vertex, respectively. If the *fp_data* facet attribute bit is set, then facet_fp_count in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each facet. If the *fp_data* vertex attribute bit is set, then vertex_fp_count in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If only one *fp_data* bit is set, the size corresponding to the other *fp_data* bit is ignored. If neither *fp_data* bit is set, fp_data_counts is ignored.

If the *2D_coords* vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

If the *edges* bit of *vert_attributes* is set, edge flags are additionally specified in the required vertex edge bitmask. This edge data is used if the pipeline state’s surface edge flag is On. For the first of three vertices that define a triangle, both edge flag bits (0x00000001 and 0x00000002) are ignored. For the second of three vertices that define a triangle, the first edge flag bit (0x00000001) indicates whether or not to draw the edge between the first and second vertices of the triangle. The second edge flag (0x00000002) is ignored. For the third of three vertices that define a triangle, the first edge flag bit indicates whether or not to draw the edge between the second and third vertices of the triangle. The second edge flag indicates whether or not to draw the edge between the third and first vertices of the triangle. If the edges vertex attribute is not set, all the edges are drawn if the pipeline state’s surface edge flag is On.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Independent Triangleswith Data.

An Independent Triangles primitive containing fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when rendered when included in a **PEXRenderOutputCommands** request.

**Encoding: Independent Triangles with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 155 | output command number |
| 2 | 5+(vc/3*d+vc*v)/4 OR 0 | output command length |
| k†1 | 6+(vc/3*d+vc*v)/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex data |
| 4 | CARD32 | number of vertices (vc) |
| vc/3*d | LISTofOPT_DATA | facet data (see OPT_DATA for d) |
| vc*v | LISTofVERTEX | vertex data (see VERTEX for v) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Independent Triangles with Data
†3) Only bits *color*, *normal*, *edges*, *2D_coords*, and *fp_data* are supported by Independent Triangles with Data

### 6.7.18 Indexed Independent Triangles with Data

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *edge_attributes:* | BITMASK_SHORT |
| *num_facets:* | CARD16 |
| *num_verts:* | CARD16 |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *per_facet_data*: | LISTofOPT_DATA |
| *per_vert_data*: | LISTofVERTEX |
| *per_edge_data*: | LISTofSWITCH |
| *connectivity*: | LISTofCARD16 |

This command causes each triangle in a set of triangles to be rendered independently. A list of independent triangles is a collection of triangles defined by reference into a modeling coordinates vertex list. The *per_vert_data* is defined in a list of length *num_verts* such that the first vertex is referenced as vertex 0. The vertices that make up each triangle are defined by the *connectivity* array which consists of *num_facets* 3-tuples of indices into the *per_vert_data* vertex list.

*facet_attributes* indicates the optional data specified for each triangle. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *fp_data*: | LISTofFLOAT |

If any of the bits in *facet_attributes* is set indicating facet data is supplied, there are *num_facets* entries of *per_facet_data* containing that facet data. Otherwise *per_facet_data* is empty.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) |
| *fp_data*: | LISTofFLOAT |

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the facet and/or vertex attributes is set, then extended data is specified with each facet and/or vertex, respectively. If the *fp_data* facet attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each facet. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If only one *fp_data* bit is set, the size corresponding to the other *fp_data* bit is ignored. If neither *fp_data* bit is set, *fp_data_counts* is ignored.

If the 2D_coords vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

*edge_attributes* indicates whether edge flags are specified for each edge. The components of the *edge_attributes* bitmask are:

*edges*: SWITCH

Edge flags are present in *per_edge_data* if the *edges* bit is set. This edge data is used if the pipeline state's *surface edge flag* is *On*. *per_edge_data* is a list of 3**num_facets* entries, each of which contains a switch indicating whether the corresponding edge is drawn. Each edge flag specifies whether the edge between vertices is to be rendered. The list corresponds to the order in which vertices and triangles are specified in *connectivity*, with the first edge flag in *per_edge_data* corresponding to the edge between the first and second vertices of the first triangle, the second edge flag corresponding to the edge between the second and third vertices of the triangle, the third edge flag corresponding to the edge between the third and first vertices of the triangle, and so on for each triangle. If the *edges* bit is not set, the *per_edge_data* list is empty and all the edges are drawn if the pipeline state's *surface edge flag* is *On*.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Indexed Independent Triangleswith Data.

An *OutputCommand* error is reported if any of the edge flags in the *per_edge_data* list is not *On* or *Off* or if any of the vertex indices in the *connectivity* array is out of range.

**Encoding: Indexed Independent Triangles with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 113 | output command number |
| 2 | 5+(*fc***d*+*vc***v*+6**fc*+*p*+3**fc***e*+*q*)/4 OR 0 | output command length |
| *k*†1 | 6+(*fc***d*+*vc***v*+6**fc*+*p*+3**fc***e*+*q*)/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | BITMASK_SHORT | edge attributes (*e*=1 if a bit is set; *e*=0 otherwise) |
| | 0x0001 *on* | |
| 2 | CARD16 | number of vertices (*vc*) |
| 2 | CARD16 | number of facets (*fc*) |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex floating-point data lists |
| *fc***d* | LISTofOPT_DATA | facet data (see OPT_DATA for *d*) |
| *vc***v* | LISTofVERTEX | vertex data (see VERTEX for *v*) |
| 6**fc* | LISTofCARD16 | connectivity data |
| *p* | | p=pad(6**fc*) |
| 3**fc***e* | LISTofSWITCH | edge data |
| *q* | | *q*=pad(3**fc***e*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Indexed Independent Triangles with Data.
†3) Only bits *color*, *normal*, *2D_coords*, and *fp_data* are supported by Indexed Independent Triangles with Data.

### 6.7.19 Marker

*points*: LISTofCOORD_3D

This command causes a set of markers to be rendered. A marker is a symbol rendered at the position given in modeling coordinates.

A marker has no geometric attributes other than its position, so transformations will not affect its size. Markers are not affected by lighting or texturing.

The Marker attribute set and marker clipping are applied to Marker.

**Encoding: Marker**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 79 | output command number |
| 2 | $1+3*fp*n/4$ OR 0 | output command length |
| $k$[†1] | $2+3*fp*n/4$ | 32-bit output command length |
| $3*fp*n$[†2] | LISTofCOORD_3D | points (length implicit) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) $n$ is the number of markers. It is derived from the output command length.

### 6.7.20 Marker 2D

5.0

*points*: LISTofCOORD_2D

This command causes a set of markers to be rendered. The command is a 2D form of the Marker output command (§6.7.19), and differs from it only by containing 2D values for the positions defined through *points*. The Z-coordinate of the positions is implicitly zero. Once these implicit values are associated with the primitive, it is rendered as a Marker primitive.

**Encoding: Marker 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 80 | output command number |
| 2 | 1+*fp***n*/2 OR 0 | output command length |
| *k*†1 | 2+*fp***n*/2 | 32-bit output command length |
| 2**fp***n*†2 | LISTofCOORD_2D | points (length implicit) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *n* is the number of markers. It is derived from the output command length.

### 6.7.21 Marker with Data

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_count*: | FP_DATA_COUNTS |
| *points*: | LISTofVERTEX |

This command causes a set of markers to be rendered. This primitive is identical to the Marker primitive except additional information can be specified for each marker position.

*vert_attributes* indicates the optional data specified with each marker position. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *fp_data*: | LISTofFLOAT |

If any of these bits is set, the corresponding data is indicated for each vertex.

If the *color* vertex attribute is set, the vertex colors are of the type specified by *color_type*. Otherwise *color_type* is ignored.

If the *fp_data* bit is set, extended data is specified with each vertex. The *vertex_fp_count* in *fp_data_count* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If the *fp_data* bit is not set, *fp_data_count* is ignored.

The Marker attribute set and marker clipping are applied to Marker with Data.

**Encoding: Marker with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 108 | output command number |
| 2 | 4+$n$*$v$/4 OR 0 | output command length |
| $k$†1 | 5+$n$*$v$/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | vertex attributes |
| 4 | CARD32 | number of markers ($n$) |
| 4 | FP_DATA_COUNTS | the number of floats in the vertex floating-point data list |
| $n$*$v$ | LISTofVERTEX | vertices (see VERTEX for $v$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bits *color* and *fp_data* are supported by Marker with Data.

### 6.7.22 Non-uniform B-spline Curve 5.0

| | |
|---|---|
| *order*: | CARD16 |
| *type*: | COORD_TYPE |
| *tmin*: | FLOAT |
| *tmax*: | FLOAT |
| *knots*: | LISTofFLOAT |
| *points*: | LISTofCOORD |

This command causes a non-uniform B-spline curve to be rendered. A non-uniform B-spline curve is a free-form curve defined as a function of a single independent variable *t*.

*order* defines the degree of the polynomials comprising the function which defines the curve.

*type* specifies whether the curve is *Rational* or *NonRational*. If *type* is *Rational*, the control points are provided as homogeneous modeling coordinates (COORD_4D), otherwise the control points are provided as non-homogeneous modeling coordinates (COORD_3D).

The parameter range values *tmin* and *tmax* specify the range over which the B-spline curve is evaluated. *Tmin* must be less than *tmax*, *tmin* must be greater than or equal to the *order*'th knot value, and *tmax* must be less than or equal to the (*k*+1-*order*)'th knot value, where *k* is the number of knots.

The list of *knots*, defined in parametric coordinate space, relates the control points to the equation which defines the curve. *knots* must form a non-decreasing sequence of numbers. The number of knots, *k*, must be equal to the number of control points plus *order*.

*points* defines the control points in modeling coordinates. The number of control points must be at least as large as *order*.

The Line and Curve attribute sets and line clipping are applied to Non-uniform B-spline Curve.

A Non-uniform B-spline Curve with an order not supported by the server is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request. *MaxNURBOrder* can be inquired with request **PEXGetImpDepConstants**.

An *OutputCommand* error is reported if *type* is not *Rational* or *NonRational*, *order* is less than one, there are fewer control points than the order, *order* is inconsistent with the number of knots and control points, *knots* is not non-decreasing, *tmin* and *tmax* are inconsistent with the knots, or a homogeneous control point has a W-coordinate that is less than or equal to zero.

**Encoding: Non-uniform B-spline Curve**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 88 | output command number |
| 2 | 4+*fp**(2+*kn*+*t**cp)/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 5+*fp**(2+*kn*+*t**cp)/4 | 32-bit output command length |
| 2 | CARD16 | order |
| 2 | COORD_TYPE | type |
| *fp* | FLOAT | tmin |
| *fp* | FLOAT | tmax |
| 4 | CARD32 | number of knots (*kn*) |
| 4 | CARD32 | number of control points (*cp*) |
| *fp*kn* | LISTofFLOAT | knots |
| $fp*t^{\dagger 2}*cp$ | LISTofCOORD | control points (COORD_3D or COORD_4D, depending on *t*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *t* = 3 if *type* is *NonRational*. *t* = 4 if *type* is *Rational*.

### 6.7.23 Non-uniform B-spline Curve with Color

| | |
|---|---|
| *shape*: | CARD16 |
| *spline_attributes*: | PRIM_DATA_ATTRIBUTES |
| *order*: | CARD16 |
| *type*: | COORD_TYPE |
| *tmin*: | FLOAT |
| *tmax*: | FLOAT |
| *knots*: | LISTofFLOAT |
| *points*: | LISTofCOORD |
| *color_spline*: | OPT_CURVE_COLORSPLINE |

This command causes a non-uniform B-spline curve primitive to be rendered. This primitive is identical to the Non-uniform B-Spline Curve primitive (§6.7.22) except additional information can be specified for the curve.

*shape* is a hint indicating the geometrical shape of the curve. A value of *UnknownNURBS* means nothing is known about the shape of the curve, and the server may determine the hint itself. The other values are: *Line*, *Circle*, *CircularArc*, *Ellipse*, *EllipticalArc*, *ParabolicArc*, and *HyperbolicArc*. The visual effect is implementation-dependent if the given *shape* does not match the curve.

Usage of *order*, *type*, *tmin, tmax, knots,* and *points* is as defined for the Non-uniform B-spline Curve primitive.

*spline_attributes* indicates the optional data specified with the curve. The components of the *spline_attributes* bitmask are:

| | |
|---|---|
| *color*: | CURVE_COLORSPLINE |

If the *color* bit of *spline_attributes* is set, *color_spline* is included to define the color distribution of the primitive's geometry. The color of any point on the primitive's geometry is determined by evaluating the color spline at the corresponding parametric value. The color spline does not have to have the same order, rationality, or knot sequence as the geometric curve. The parameter range of the color spline must wholly include the bound parameter range of the geometric spline. Each control point's components correspond to a direct color. If the color spline is rational, the direct color components include a homogeneous coordinate. The color components may be outside the range normally associated with the color type. The number of color spline control points must be at least as large as the color spline's order. The number of knots must be equal to the number of control points plus the order. The knot values sequence must be non-decreasing.

The Line and Curve attribute sets and line clipping are applied to Non-uniform B-spline Curve with Color.

A Non-uniform B-spline Curve with Color with a geometric or color spline order not supported by the server is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request. *MaxNURBOrder* can be inquired with request **PEXGetImpDepConstants**.

An *OutputCommand* error is reported for the geometric or color spline, if *type* is not *Rational* or *NonRational*, *order* is less than one, there are fewer control points than the order, *order* is inconsistent with the number of knots and control points, *knots* is not non-decreasing, *tmin* and *tmax* are inconsistent with the knots, or a homogeneous control point has a W-coordinate that is less than or equal to zero. An *OutputCommand* error is also reported if the *shape* parameter is invalid or if the color spline's color type is *Indexed*.

**Encoding: Non-uniform B-spline Curve with Color**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | 109 | | output command number |
| 2 | 5+((2+*kn*+*t***cp)***fp*+*s*)/4 OR 0 | | output command length |
| $k^{\dagger 1}$ | 6+((2+*kn*+*t***cp)***fp*+*s*)/4 | | 32-bit output command length |
| 2 | CARD16 | | shape |
| | 0 | *UnknownNURB* | |
| | 1 | *Circle* | |
| | 2 | *CircularArc* | |
| | 4 | *Ellipse* | |
| | 5 | *EllipticalArc* | |
| | 7 | *Line* | |
| | 8 | *ParabolicArc* | |
| | 9 | *HyperbolicArc* | |
| 2 | PRIM_DATA_ATTRIBUTES†2 | | spline attributes |
| 2 | CARD16 | | order |
| 2 | COORD_TYPE | | type |
| *fp* | FLOAT | | tmin |
| *fp* | FLOAT | | tmax |
| 4 | CARD32 | | number of knots (*kn*) |
| 4 | CARD32 | | number of control points (*cp*) |
| *fp***kn* | LISTofFLOAT | | knots |
| $fp*t^{\dagger 3}*cp$ | LISTofCOORD | | control points (COORD_3D or COORD_4D, depending on *t*) |
| *s* | OPT_CURVE_COLORSPLINE | | color spline (see §6.7.23.1 for *s*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bit *color* is supported by Non-uniform B-spline Curve with Color.
†3) *t* = 3 if *type* is *NonRational*. *t* = 4 if *type* is *Rational*.

#### 6.7.23.1 Type OPT_CURVE_COLORSPLINE

OPT_CURVE_COLORSPLINE :
optional [color_type : COLOR_TYPE,
order : CARD16,
type : COORD_TYPE,
knots : LISTofFLOAT,
points : LISTofSPLINE_COLOR]

**Encoding: OPT_CURVE_COLORSPLINE**

| # of Bytes[†1] | Data Type or Value | Description |
|---|---|---|
| 2 | COLOR_TYPE | color type |
| 2 | CARD16 | order |
| 2 | COORD_TYPE | type |
| 2 | | unused |
| 4 | CARD32 | number of knots (*kn*) |
| 4 | CARD32 | number of points (*cp*) |
| *kn*fp* | LISTofFLOAT | knots |
| *cp*c* | LISTofSPLINE_COLOR | points (see SPLINE_COLOR for *c*) |

†1) The fields are optionally specified together. Size *s* of an OPT_CURVE_COLORSPLINE is as follows. The base size is 0. The size increases to $kn*fp+cp*c+16$ if the color spline is present.

### 6.7.24 Non-uniform B-spline Surface

5.0

| | |
|---|---|
| *type*: | COORD_TYPE |
| *u_order*: | CARD16 |
| *v_order*: | CARD16 |
| *u_knots*: | LISTofFLOAT |
| *v_knots*: | LISTofFLOAT |
| *mpts*: | CARD16 |
| *npts*: | CARD16 |
| *points*: | LISTofCOORD |
| *trim_curves*: | LISTofLISTofTRIM_CURVE |

This command causes a non-uniform B-spline surface to be rendered. A non-uniform B-spline surface is a free-form surface defined as a function of two independent variables *u* and *v*.

*u_order* and *v_order* define the degree of the polynomials comprising the function which defines the surface.

*type* specifies whether the surface is *Rational* or *NonRational*. If *type* is *Rational*, the control points are provided as homogeneous modeling coordinates (COORD_4D), otherwise the control points are provided as non-homogeneous modeling coordinates (COORD_3D).

The list of *u_knots* and *v_knots* relate the control points to the function which defines the surface. *u_knots* and *v_knots* must each form a non-decreasing sequence of numbers. The minimum and maximum knot values in *u_knots* define the range over which the surface is evaluated in the *u* direction, and the minimum and maximum knot values in *v_knots* define the range over which the surface is evaluated in the *v* direction. The number of *u* knots, *uk*, must be equal to the number of control points in the *u* direction plus *u_order*. Likewise the number of *v* knots, *vk*, must be equal to the number of control points in the *v* direction plus *v_order*.

*points* defines the array of control points in modeling coordinates. *mpts* defines the number of control points in the *u* direction, and *npts* defines the number of control points in the *v* direction. *mpts* must be at least as large as *u_order*, and *npts* must be at least as large as *v_order*. The control points are stored in row major order (i.e., the column number varies fastest as the points are stored in the array) and the rows increase in the direction of increasing *v*.

A list of trimming loops may be specified to further restrict the parametric coordinate space region over which the surface is evaluated. Each trimming loop is defined by one or more B-spline trim curves that are connected head-to-tail. Each trim loop must be explicitly closed with the tail of its last trim curve joining the head of its first trim curve. If there is a floating-point inaccuracy in loop closure or in trim curve head-to-tail connectivity, closure or connectivity will be assumed. Each trim curve is parameterized independently so each trim curve and the surface may have a differing order, rationality, and knot sequence. However the trim curves are defined in the parameter space of the surface and may not go outside the parameter space of the surface.

Trimming loops define the region of the surface that is to be rendered based on the following two rules: (1) a point is in the portion of the surface to be rendered if any ray projected from it to infinity has an odd number of intersections with trimming loops, and (2) traveling in the direction of a trimming loop, the portion of the surface to be trimmed away should be on the right and the portion to be retained should be on the left. In other words, a loop defined in counterclockwise order causes the interior of the loop to be retained and the exterior to be clipped away. A clockwise loop causes the exterior of the loop to be retained and the interior to be trimmed away. If loops are nested, they must alternate in direction. In all cases, the outermost loop must be counterclockwise. No trim curve may intersect itself and no trimming loop intersect itself or any other trimming loop. Trimming loops that do not obey these rules will result in an implementation-dependent rendering.

Usage of the trim curve *order*, *type, tmin, tmax,* and *knots* fields is as defined for the Non-uniform B-Spline Curve primitive (§6.7.22). Each control point's components correspond to a parameter space point (*u,v*) with an additional homogeneous coordinate if the trim curve is rational. The *trim_curve_approximation_method* field specifies the smoothing method to be used when rendering the surface edge formed by the trim curve. The defined trim curve approximation method values are defined in Table 6-18 .

**Table 6-18 . Enumerated Type *TrimCurveApproxMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *(implementation-dependent)* | The surface edge formed by the trim curve is rendered in an implementation-dependent manner. 5.0 |
| 2 | *ConstantBetweenKnots* | The trim curve is tessellated with equal parametric increments between successive pairs of knots. The *tolerance* field of the trim curve controls the tessellation. If the tolerance does not define an integer value, it is truncated and only the integer portion is used. If *tolerance* is less than or equal to zero, the trim curve will be evaluated only at the parameter limits and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, the trim curve will be evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by *tolerance* between each pair of knots. 5.0 |

Each trim curve has a visibility flag which allows individual control over whether the edge is rendered. When no trimming loops are defined, just the outer rectangular parameter limits of the surface define the edges. This edge data is used if the pipeline state's *surface edge flag* is *On*.

The General Surface, Styled and Shaded Surface, Edge, Curved Surface, and Parametric Surface attribute sets and surface clipping are applied to Non-uniform B-spline Surface.

A Non-uniform B-spline Surface with a surface or trim curve order not supported by the server is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request. *MaxNURBOrder* and *MaxTrimCurveOrder* can be inquired with request **PEXGetImpDepConstants**.

An *OutputCommand* error is reported for the surface or a trim curve, if the type is not *Rational* or *NonRational*, a surface order is less than one or a trim curve order less than two, there are fewer control points than the order, the order is inconsistent with the number of knots and control points, the knots are not non-decreasing, a homogeneous control point has a W-coordinate that is less than or equal to zero, or a trim curve's parameter range is inconsistent with its knots.

**Encoding: Non-uniform B-spline Surface**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 98 | output command number |
| 2 | 7+*fp**(*uk*+*vk*+*m***n***t*)/4+*l*+*c*/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 8+*fp**(*uk*+*vk*+*m***n***t*)/4+*l*+*c*/4 | 32-bit output command length |
| 2 | COORD_TYPE | type |
| 2 | CARD16 | *u* order |
| 2 | CARD16 | *v* order |
| 2 | | unused |
| 4 | CARD32 | number of *u* knots (*uk*) |
| 4 | CARD32 | number of *v* knots (*vk*) |
| 2 | CARD16 | number of points in the *u* direction (*m*) |
| 2 | CARD16 | number of points in the *v* direction (*n*) |
| 4 | CARD32 | number of trimming loops (*l*) |
| *uk*fp* | LISTofFLOAT | *u* knots |
| *vk*fp* | LISTofFLOAT | *v* knots |
| *m*n*fp*t* | LISTofCOORD | control points (COORD_3D or COORD_4D depending on $t^{\dagger 2}$) |
| *4*l+c* | LISTofLISTofTRIM_CURVE | trim curves (see TRIM_CURVE for *c*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *t* = 3 if *type* is *NonRational*. *t* = 4 if *type* is *Rational*.

### 6.7.25 Non-uniform B-spline Surface with Data

| | |
|---|---|
| *shape*: | CARD16 |
| *spline_attributes*: | PRIM_DATA_ATTRIBUTES |
| *type*: | COORD_TYPE |
| *u_order*: | CARD16 |
| *v_order*: | CARD16 |
| *u_knots*: | LISTofFLOAT |
| *v_knots*: | LISTofFLOAT |
| *mpts*: | CARD16 |
| *npts*: | CARD16 |
| *points*: | LISTofCOORD |
| *trim_curves*: | LISTofLISTofTRIM_CURVE |
| *color_spline*: | OPT_SURFACE_COLORSPLINE |
| *data_splines*: | OPT_LISTofSURFACE_DATASPLINE |

This command causes a non-uniform B-spline surface primitive to be rendered. This primitive is identical to the Non-uniform B-Spline Surface primitive (§ 6.7.24) except additional information can be specified for the surface.

*shape* is a hint indicating the geometrical shape of the surface. A value of *UnknownNURBS* means nothing is known about the shape of the surface. The other values are: *Plane, Cone, Sphere, Torus, RightCircularCylinder, SurfaceOfRevolution, TabulatedCylinder, RuledSurface*, and *GeneralQuadric*. The visual effect is implementation-dependent if the given *shape* does not match the surface.

Usage of *type, u_order*, *v_order, u_knots, v_knots, mpts, npts, points* and *trim_curves* is as defined for the Non-uniform B-spline Surface primitive.

*spline_attributes* indicates the optional data specified with the surface. The components of the *spline_attributes* bitmask are:

| | |
|---|---|
| *color*: | SURFACE_COLORSPLINE |
| *fp_data:* | LISTofSURFACE_DATASPLINE |

If any of these bits is 1, the corresponding attribute spline surface(s) must be included with the primitive.

The color and data attribute splines do not have to have the same orders, rationality, or knot sequences as the geometric surface. The number of attribute spline control points in each parameter must be at least as large as the spline's order in that parameter. The number of knots in each parameter must be equal to the number of control points plus the order in that direction. The parameter ranges of the attribute spline must wholly include the bound parameter ranges of the geometric spline, or, if trimming loops are specified for the geometric surface, the parameter range must wholly contain the parameter range defined by the trimming loops.

If the *color* bit of *spline_attributes* is set, *color_spline* is included to define the color distribution of the primitive's geometry. The color of any point on the primitive's geometry is determined by evaluating the color spline at the corresponding parametric values. Each control point's components correspond to a direct color. If the color spline is rational, the direct color components include a homogeneous coordinate. The color components may be outside the range normally associated with the color type.

If the *data* bit of *spline_attributes* is set, *data_splines* is included to define the distribution over the primitive's geometry of each floating-point value. Each data spline may represent the distribution of any number of floating-point values. A specific floating-point value is determined by evaluating the appropriate data spline at the corresponding parametric values. Each of the control point's components correspond to a specific floating-point value. If a data spline is rational, the control point additionally includes a homogeneous coordinate. Because there are multiple data splines and each data spline may represent multiple floating-point values, functionalities referencing the floating-point values must use two indices to indicate a specific floating-point value distribution. If such a functionality only provides one index (as is required for all other primitives with floating-point data), it will be assumed this index selects which data spline is to be used. The first floating-point value of that spline is then implicitly selected.

The General Surface, Styled and Shaded Surface, Edge, Curved Surface, Parametric Surface, and Data Mapping attribute sets and surface clipping are applied to Non-uniform B-spline Surface with Data.

A Non-uniform B-spline Surface with Data with a geometric surface, trim curve, color surface, or data surface order not supported by the server is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request. *MaxNURBOrder* and *MaxTrimCurveOrder* can be inquired with request **PEXGetImpDepConstants**.

An *OutputCommand* error is reported for the geometric or attribute surfaces, or a trim curve, if the type is not *Rational* or *NonRational*, a surface order is less than one or a trim curve order less than two, there are fewer control points than the order, the order is inconsistent with the number of knots and control points, the knots are not non-decreasing, a homogeneous control point has a W-coordinate that is less than or equal to zero, or a trim curve's parameter range is inconsistent with its knots. An *OutputCommand* error is also reported if the *shape* parameter is invalid or if the color spline's color type is *Indexed*.

**Encoding: Non-uniform B-spline Surface with Data**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | 110 | | output command number |
| 2 | 9+(*uk*+*vk*+*m***n***t*)**fp*/4+*l*+(*c*+*s*+*d***b*)/4 OR 0 | | output command length |
| *k*†1 | 10+(*uk*+*vk*+*m***n***t*)**fp*/4+*l*+(*c*+*s*+*d***b*)/4 | | 32-bit output command length |
| 2 | CARD16 | | shape |
| | 0 | *UnknownNURB* | |
| | 1 | *Plane* | |
| | 2 | *Cone* | |
| | 3 | *Sphere* | |
| | 4 | *Torus* | |
| | 5 | *RightCircularCylinder* | |
| | 6 | *SurfaceOfRevolution* | |
| | 7 | *TabulatedCylinder* | |
| | 8 | *RuledSurface* | |
| | 9 | *GeneralQuadric* | |
| 2 | PRIM_DATA_ATTRIBUTES†2 | | spline attributes |
| 2 | COORD_TYPE | | type |
| 2 | CARD16 | | *u* order |
| 2 | CARD16 | | *v* order |
| 2 | | | unused |
| 4 | CARD32 | | number of *u* knots (*uk*) |
| 4 | CARD32 | | number of *v* knots (*vk*) |
| 2 | CARD16 | | number of points in the *u* direction (*m*) |
| 2 | CARD16 | | number of points in the *v* direction (*n*) |
| 4 | CARD32 | | number of trimming loops (*l*) |
| 4 | CARD32 | | number of data splines (*d*) |
| *uk***fp* | LISTofFLOAT | | *u* knots |
| *vk***fp* | LISTofFLOAT | | *v* knots |
| *m***n***fp***t* | LISTofCOORD | | control points (COORD_3D or COORD_4D depending on *t*†3) |
| 4**l*+*c* | LISTofLISTofTRIM_CURVE | | trim curves (see TRIM_CURVE for *c*) |
| *s* | OPT_SURFACE_COLORSPLINE | | color spline (see §6.7.25.1 for *s*) |
| *d***b* | OPT_LISTofSURFACE_DATASPLINE | | data splines (see §6.7.25.2 for *b*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color* and *fp_data* are supported by Non-uniform B-spline Surface with Data.
†3) *t* = 3 if *type* is *NonRational*. *t* = 4 if *type* is *Rational*.

#### 6.7.25.1 Type OPT_SURFACE_COLORSPLINE

OPT_SURFACE_COLORSPLINE :

optional [color_type : COLOR_TYPE,
type : COORD_TYPE,
u_order, v_order : CARD16,
u_knots, v_knots : LISTofFLOAT,
mpts, npts : CARD16,
points : LISTofSPLINE_COLOR]

**Encoding: OPT_SURFACE_COLORSPLINE**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| 2 | COLOR_TYPE | color type |
| 2 | COORD_TYPE | type |
| 2 | CARD16 | u order |
| 2 | CARD16 | v order |
| 4 | CARD32 | number of u knots (*uk*) |
| 4 | CARD32 | number of v knots (*vk*) |
| 4 | CARD32 | number of points in the u direction (*m*) |
| 4 | CARD32 | number of points in the v direction (*n*) |
| *uk*fp* | LISTofFLOAT | u knots |
| *vk*fp* | LISTofFLOAT | v knots |
| *m*n*c* | LISTofSPLINE_COLOR | points (see SPLINE_COLOR for *c*) |

†1) The fields are optionally specified together. Size *s* of an OPT_SURFACE_COLORSPLINE is as follows. The base size is 0. The size increases to *(uk+vk)*fp+m*n*c+24* if the color spline is present.

#### 6.7.25.2 Type OPT_LISTofSURFACE_DATASPLINE

OPT_LISTofSURFACE_DATASPLINE :
optional LISTofSURFACE_DATASPLINE

SURFACE_DATASPLINE : [n_values : CARD32,
type : COORD_TYPE,
u_order, v_order : CARD16,
u_knots, v_knots : LISTofFLOAT,
mpts, npts : CARD16,
points : LISTofFLOAT]

**Encoding: SURFACE_DATASPLINE**

| # of Bytes†1 | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of floating-point values (*f*) |
| 2 | COORD_TYPE | type |
| 2 | CARD16 | u order |
| 2 | CARD16 | v order |
| 2 | | unused |
| 4 | CARD32 | number of u knots (*uk*) |
| 4 | CARD32 | number of v knots (*vk*) |
| 4 | CARD32 | number of points in the u direction (*m*) |
| 4 | CARD32 | number of points in the v direction (*n*) |
| *uk**fp* | LISTofFLOAT | u knots |
| *vk**fp* | LISTofFLOAT | v knots |
| *m**n**(t*†2*+ f)**fp* | LISTofFLOAT | points |

†1) The size *b* of a SURFACE_DATASPLINE is (*uk*+*vk*+(*f*+*t*)**m***n*)**fp*+28
†2) *t* is 0 if the data spline is *NonRational*, else *t* is 1 because the data spline is *Rational*.

### 6.7.26 Polyline

*vertices*: LISTofCOORD_3D

This command causes a polyline to be rendered. A polyline is a list of vertices defined in modeling coordinates with vertex $i$ connected to vertex $i+1$ by a line segment. The last vertex is *not* connected to the first.

Line primitives have no geometric width, only length, so transformations will affect only their displayed length.

The Line and Shaded Line attribute sets and line clipping are applied to Polyline.

A polyline containing fewer than two vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

#### Encoding: Polyline

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 85 | output command number |
| 2 | 1+3**fp***n*/4 OR 0 | output command length |
| *k*[†1] | 2+3**fp***n*/4 | 32-bit output command length |
| 3**fp***n*[†2] | LISTofCOORD_3D | vertices (length implicit) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *n* is the number of vertices in the polyline. It is derived from the output command length.

### 6.7.27 Polyline 2D

5.0

*vertices*: LISTofCOORD_2D

This output command causes a polyline to be rendered. The command is a 2D form of the Polyline output command (§6.7.26), and differs from it only by containing 2D values for the coordinates defined through *vertices*. The Z-coordinate of the vertices is implicitly 0. Once these implicit values are associated with the primitive, it is rendered as a Polyline primitive.

**Encoding: Polyline 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 86 | output command number |
| 2 | 1+*fp***n*/2 OR 0 | output command length |
| $k^{\dagger 1}$ | 2+*fp***n*/2 | 32-bit output command length |
| 2**fp**$n^{\dagger 2}$ | LISTofCOORD_2D | vertices (length implicit) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *n* is the number of vertices in the polyline. It is derived from the output command length.

### 6.7.28 Polyline Set with Data (Version 5.0) 5.0

*color_type*: COLOR_TYPE
*vert_attributes*: PRIM_DATA_ATTRIBUTES
*vertices*: LISTofLISTofVERTEX

This command causes a set of disjoint polylines to be rendered. This primitive is similar to the Polyline primitive (§6.7.26) except this primitive allows the optional specification of multiple polylines and vertex colors. The Polyline Set with Data (V5.2) primitive (§6.7.29) is exactly like this primitive except it additionally allows the optional specification of a move/draw flag and a floating-point data list with each vertex, and 2D or 3D vertices.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

*color*: COLOR 5.0

If any of these bits is set, the corresponding data is indicated for each vertex.

If the *color* vertex attribute is set, the vertex colors are of the type specified by *color_type*. Otherwise *color_type* is ignored.

The Line and Shaded Line attribute sets and line clipping are applied to Polyline Set with Data (V5.0).

A polyline set with a polyline containing fewer than two vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Polyline Set with Data (Version 5.0)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 87 | output command number |
| 2 | $3+n+m*v/4$ OR 0 | output command length |
| $k$[†1] | $4+n+m*v/4$ | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES[†2] | vertex attributes |
| 4 | CARD32 | number of polylines ($n$) |
| $4*n+m*v$ | LISTofLISTofVERTEX | vertices (see VERTEX for $v$[†3]) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bit *color* are supported by Polyline Set with Data (V5.0).
†3) $m$ is the total number of vertices which define the polyline set. In the LISTofLISTofVERTEX, each polyline is defined by a CARD32 value indicating how many vertices it contains followed by the vertices and their data (of type VERTEX).

### 6.7.29 Polyline Set with Data (Version 5.2)

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_count*: | FP_DATA_COUNTS |
| *vertices*: | LISTofLISTofVERTEX |

This command causes a set of disjoint polylines to be rendered. This primitive is similar to the Polyline primitive except this primitive allows the optional specification of multiple polylines, a color, move/draw flag, and a floating-point data list for each vertex, and 2D or 3D vertices. The Polyline Set with Data (V5.0) primitive (§6.7.28) is exactly like this primitive except it does not support vertex move/draw flags and floating-point data lists, and 2D vertices.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *edges*: | BITMASK |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) |
| *fp_data*: | LISTofFLOAT |

If any of these bits is set, the corresponding data is indicated for each vertex.

If the *color* vertex attribute is set, the vertex colors are of the type specified by *color_type*. Otherwise *color_type* is ignored.

If the *edge* vertex attribute is set, a move/draw control flag is specified in the first bit (0x00000001) of the edge bitmask of each vertex. Using move/draw control flags allows a single polyline to have disjoint components. Therefore the same disjoint lines can be defined with Polyline Set with Data (V5.2) by using multiple polylines without move/draw control flags or by using a single polyline with move/draw control flags. If the move/draw control flag is set to 1 for vertex *i*, a line is drawn between vertex *i-1* and vertex *i*. If the move/draw control flag is not set, vertex *i* is the first vertex of the next line. The move/draw control flag of the first vertex of each polyline will be ignored.

If the *2D_coords* bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

If the *fp_data* bit is set, extended data is specified with each vertex. The *vertex_fp_count* in *fp_data_count* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If the *fp_data* bit is not set, *fp_data_count* is ignored. The *facet_fp_count* field of *fp_data_count* is always ignored.

The Line and Shaded Line attribute sets and line clipping are applied to Polyline Set with Data (V5.2).

A polyline set with a polyline containing fewer than two vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Polyline Set with Data (Version 5.2)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 146 | output command number |
| 2 | $4+n+m*v/4$ OR 0 | output command length |
| $k^{\dagger 1}$ | $5+n+m*v/4$ | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | vertex attributes |
| 4 | CARD32 | number of polylines ($n$) |
| 4 | FP_DATA_COUNTS | the number of floats in the vertex floating-point data list |
| $4*n+m*v$ | LISTofLISTofVERTEX | vertices (see VERTEX for $v^{\dagger 3}$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)

†2) Only bits *color*, *edges*, *2D_coords* and *fp_data* are supported by Polyline Set with Data (V5.2).

†3) $m$ is the total number of vertices which define the polyline set. In the LISTofLISTofVERTEX, each polyline is defined by a CARD32 value indicating how many vertices it contains followed by the vertices and their data (of type VERTEX).

### 6.7.30 Quadrilateral Mesh with Data (Version 5.0)

*shape*: SHAPE
*color_type*: COLOR_TYPE
*m_pts*: CARD16
*n_pts*: CARD16
*facet_attributes*: PRIM_DATA_ATTRIBUTES
*vert_attributes*: PRIM_DATA_ATTRIBUTES
*facet_data*: LISTofOPT_DATA
*vertices*: LISTofVERTEX

This primitive is identical to the Quadrilateral Mesh with Data (V5.2) primitive (§6.7.31) except this primitive does not allow facet and floating-point data lists, vertex individual edge flags and floating-point data lists, and 2D vertices.

This command causes each quadrilateral in a mesh of quadrilaterals to be rendered independently. A quad mesh is a grid of connected four-sided polygons. The surface is created from the *vertices* which represent a vertex array stored in row major order (i.e., the column number varies fastest as vertices are stored in the array). The (*i*th, *j*th), (*i+1*th, *j*th), (*i+1*th, *j+1*th), and (*i*th, *j+1*th) vertices are connected to create a single facet. Adjacent vertices are interconnected until the entire facet network is processed. *m_pts* is the number of columns in the vertex array and *n_pts* is the number of rows. Therefore there are *m_pts* * *n_pts* entries in *vertices,* and (*m_pts* - *1*) * (*n_pts* - *1*) entries in *facet_data* if any facet attributes are defined.

The *shape* parameter is passed as a hint as to the type of quadrilaterals that comprise the primitive. A shape hint of *Unknown* means the client is not supplying any information about the possible complexity of individual quadrilaterals. A shape of *Complex* means that some quadrilaterals may have edges that self-intersect. A shape of *Nonconvex* means that none of the quadrilaterals have edges that self-intersect, but some may not be wholly convex. *Convex* means that all of the interior angles of all of the quadrilaterals are convex. Quadrilaterals that are of a simpler complexity than indicated by the shape hint are drawn correctly. Quadrilaterals that are of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner.

*facet_attributes* indicates the optional data specified for each facet. The components of the *facet_attributes* bitmask are:

*color*: COLOR
*normal*: VECTOR_3D

If any of the attribute bits is set, the corresponding attributes are present in *facet_data*.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

*color*: COLOR
*normal*: VECTOR_3D

If any of the attribute bits is set, the corresponding data is indicated for each vertex.

If facet or vertex colors are specified, they are of the type indicated by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Quadrilateral Mesh with Data (V5.0).

It is allowable for the boundary of a single facet to not reside in a single plane. The treatment of the vertex attributes in this case is implementation-dependent.

A quad mesh with either *m_pts* or *n_pts* less than two is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

An *OutputCommand* error is reported if the *shape* parameter is invalid.

**Encoding: Quadrilateral Mesh with Data (Version 5.0)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 96 | output command number |
| 2 | $4+(d*(m-1)*(n-1)+m*n*v)/4$ OR 0 | output command length |
| $k^{\dagger 1}$ | $5+(d*(m-1)*(n-1)+m*n*v)/4$ | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | CARD16 | number of points in m direction (*m*) |
| 2 | CARD16 | number of points in n direction (*n*) |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | SHAPE | shape |
| $(m-1)*(n-1)*d$ | LISTofOPT_DATA | facet data (see OPT_DATA for *d*) |
| $m*n*v$ | LISTofVERTEX | vertices (see VERTEX for *v*) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bits *color* and *normal* are supported by Quadrilateral Mesh with Data (V5.0).
†3) Only bits *color* and *normal* are supported by Quadrilateral Mesh with Data (V5.0).

### 6.7.31 Quadrilateral Mesh with Data (Version 5.2) 5.2

| | |
|---|---|
| *shape*: | SHAPE |
| *color_type*: | COLOR_TYPE |
| *m_pts*: | CARD16 |
| *n_pts*: | CARD16 |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *facet_data*: | LISTofOPT_DATA |
| *vertices*: | LISTofVERTEX |

This primitive is identical to the Quadrilateral Mesh with Data (V5.0) primitive (§6.7.30) except this primitive allows facet floating-point data lists, vertex individual edge flags and floating-point data lists, and 2D vertices to be optionally specified.

This command causes each quadrilateral in a mesh of quadrilaterals to be rendered independently. A quad mesh is a grid of connected four-sided polygons. The surface is created from the *vertices* which represent a vertex array stored in row major order (i.e., the column number varies fastest as vertices are stored in the array). The (*i*th, *j*th), (*i+1*th, *j*th), (*i+1*th, *j+1*th), and (*i*th, *j+1*th) vertices are connected to create a single facet. Adjacent vertices are interconnected until the entire facet network is processed. *m_pts* is the number of columns in the vertex array and *n_pts* is the number of rows. Therefore there are *m_pts* * *n_pts* entries in *vertices,* and (*m_pts* - *1*) * (*n_pts* - *1*) entries in *facet_data* if any facet attributes are defined.

The *shape* parameter is passed as a hint as to the type of quadrilaterals that comprise the primitive. A shape hint of *Unknown* means the client is not supplying any information about the possible complexity of individual quadrilaterals. A shape of *Complex* means that some quadrilaterals may have edges that self-intersect. A shape of *Nonconvex* means that none of the quadrilaterals have edges that self-intersect, but some may not be wholly convex. *Convex* means that all of the interior angles of all of the quadrilaterals are convex. Quadrilaterals that are of a simpler complexity than indicated by the shape hint are drawn correctly. Quadrilaterals that are of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner.

*facet_attributes* indicates the optional data specified for each facet. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of the attribute bits is set, the corresponding attributes are present in *facet_data*.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *edges*: | BITMASK 5.2 |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) 5.2 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of the attribute bits is set, the corresponding data is indicated for each vertex.

If facet or vertex colors are specified, they are of the type indicated by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the facet attributes and/or vertex attributes is set, then extended data is specified with each facet and/or vertex, respectively. If the *fp_data* facet attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each facet. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. Each field in *fp_data_counts* is ignored if its corresponding *fp_data* bit is not set.

If the *edges* bit of the vertex attributes is set, two surface edge flags are specified with each vertex. This edge data is used if the pipeline sate's *surface edge flag* is *On.* For the (*i*th, *j*th) vertex, the first edge flag bit (0x00000001) of the vertex's edge bitmask indicates whether to draw the edge between the (*i-1*th, *j*th) and (*i*th, *j*th) vertices, and the second bit (0x00000002) of the edge bitmask indicates whether to draw the edge between the (*i*th, *j-1*th) and (*i*th, *j*th) vertices. The first edge flag is ignored for all vertices where $i = 1$ and the second edge flag is ignored for all vertices where $j = 1$. If the *edges* vertex attribute is not set, all the edges are drawn if the pipeline state's *surface edge flag* is *On.*

If the *2D_coords* vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Quadrilateral Mesh with Data (V5.2).

It is allowable for the boundary of a single facet to not reside in a single plane. The treatment of the vertex attributes in this case is implementation-dependent.

A quad mesh with either *m_pts* or *n_pts* less than two is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

An *OutputCommand* error is reported if the *shape* parameter is invalid.

**Encoding: Quadrilateral Mesh with Data (Version 5.2)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 150 | output command number |
| 2 | 5+(*d**(*m*-1)*(*n*-1)+*m**n*v*)/4 OR 0 | output command length |
| *k*†1 | 6+(*d**(*m*-1)*(*n*-1)+*m**n*v*)/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | CARD16 | number of points in m direction (*m*) |
| 2 | CARD16 | number of points in n direction (*n*) |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | SHAPE | shape |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex floating-point data lists |
| (*m*-1)*(*n*-1)**d* | LISTofOPT_DATA | facet data (see OPT_DATA for *d*) |
| *m**n*v* | LISTofVERTEX | vertices (see VERTEX for *v*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Quadrilateral Mesh with Data (V5.2).
†3) Only bits *color*, *normal*, *edges*, *2D_coords*, and *fp_data* are supported by Quadrilateral Mesh with Data (V5.2).

### 6.7.32 Set of Fill Area Sets with Data (Version 5.0)

| | |
|---|---|
| *shape*: | SHAPE |
| *color_type*: | COLOR_TYPE |
| *fas_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *edge_attributes*: | BITMASK_SHORT |
| *contour_hint*: | CONTOUR |
| *contours_all_1*: | BOOLEAN |
| *num_fas*: | CARD16 |
| *num_verts*: | CARD16 |
| *num_edges*: | CARD16 |
| *num_contours*: | CARD16 |
| *per_fas_data*: | LISTofOPT_DATA |
| *per_vert_data*: | LISTofVERTEX |
| *per_edge_data*: | LISTofSWITCH |
| *contours*: | LISTofLISTofLISTofCARD16 |

This primitive is identical to the Set of Fill Area Sets with Data (V5.2) primitive (§6.7.33) except this primitive does not allow facet and vertex floating-point data lists and 2D vertices.

This command causes each fill area set in a set of fill area sets to be rendered independently. A set of fill area sets is a collection of not necessarily coplanar Fill Area Sets with Data (§6.7.13). The vertices are defined in a single list which is referenced by the definition of each fill area set. This allows vertex data to be defined once and then shared.

*num_verts* defines the number of vertices defined in the list of vertices, *per_vert_data*. The connectivity of these vertices is defined through *contours.* Each fill area set is defined by one or more lists of vertices. Each list forms one contour of the primitive. *num_contours* indicates the total number of contours in the primitive. *contours* indicates the number of contours in each fill area set, the number of vertices in each contour, and which vertices define each contour. The vertices are defined by referencing into *per_vert_data*, where the first vertex is referenced by index 0.

*contours_all_1* is *True* if there is only one contour in each fill area set. The results are implementation-dependent if *contours_all_1* is set to *True* when there are no contours in any or all of the fill area sets, or if there are more than one contour in any of the fill area sets.

*fas_attributes* indicates the optional data specified for each fill area set. The components of the *fas_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |

If any of these bits is 1, the corresponding data is available in the *num_fas* entries of *per_fas_data*. Otherwise *per_fas_data* is empty. Be aware if even a fill area set has zero contours defined, PEX still expects *per_fas_data* to contain an entry for that fill area set.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |

If any of these bits is set, the corresponding data is indicated for each vertex.

If fill area set and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for fill area set and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

*edge_attributes* indicates the optional data specified for each edge. The components of the *edge_attributes* bitmask are:

| | |
|---|---|
| *edges*: | SWITCH |

Edge flags are present in *per_edge_data* if the *edges* bit is set. This edge data is used if the pipeline state's *surface edge flag* is *On*. *per_edge_data* is a list of *num_edges* entries. The list corresponds to the order in which vertices are specified in *contours*, even if the vertices form a degenerate contour which is not drawn. The first edge flag in *per_edge_data* corresponds to the edge between the first and second vertices of the first contour of the first fill area set, and so on. Each entry of *per_edge_data* contains a switch indicating whether an edge is drawn from the corresponding vertex to the next vertex. The edge flag corresponding with the last vertex of each contour indicates whether an edge is drawn from the last vertex to the first vertex of the contour. If the *edges* bit is not set, the *per edge data* list is empty and all the edges are drawn if the pipeline state's *surface edge flag* is *On*.

Usage of *shape* and *contour_hint* is as defined for the Fill Area Set primitive (§6.7.11).

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Set of Fill Area Sets with Data (V5.0).

An *OutputCommand* error is reported if any of the edge flags in the *per_edge_data* list is not *On* or *Off*, or if any of the *shape, contour_hint,* or *contours_all_1* parameters is invalid, or if any of the vertex indices in the *contours* array is out of range, or if there are not as many edge flags as they are vertex indices in the lists in the contours array.

**Encoding: Set of Fill Area Sets with Data (Version 5.0)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 97 | output command number |
| 2 | 0 OR 6+ (*fc***d*+*vc***v*+*ec***e*+*p1*+2**fc*+2**cc*+2**ec*+*p2*)/ 4 | output command length |
| *k*†1 | 7+ (*fc***d*+*vc***v*+*ec***e*+*p1*+2**fc*+2**cc*+2**ec*+*p2*)/ 4 | 32-bit output command length |
| 2 | SHAPE | shape |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | fill area set attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | BITMASK_SHORT | edge attributes (call this value *e*) |
| | 0x0001 *individual_edges* | |
| 1 | CONTOUR | contour hint |
| 1 | BOOLEAN | all contour counts = 1 |
| 2 | CARD16 | number of fill area sets (*fc*) |
| 2 | CARD16 | number of vertices (*vc*) |
| 2 | CARD16 | number of edges (sum of all contour counts) (*ec*) |
| 2 | CARD16 | number of contours (*cc*) |
| *fc***d* | LISTofOPT_DATA | fill area set data (see OPT_DATA for *d*) |
| *vc***v* | LISTofVERTEX | vertex data (see VERTEX for *v*) |
| *ec***e* | LISTofSWITCH | edge data |
| *p1* | | pad(*ec***e*) |
| 2**fc*+2**cc*+ 2**ec* | LISTofLISTofLISTofCARD16 | connectivity data (see §6.7.32.1) |
| *p2* | | *p*=pad(2**fc*+2**cc*+2**ec*) |

†1) *k* = ((output command length == 0) : 4 : 0)
†2) Only bits *color* and *normal* are supported by Set of Fill Area Sets with Data (V5.0).
†3) Only bits *color* and *normal* are supported by Set of Fill Area Sets with Data (V5.0).

### 6.7.32.1 Connectivity Data

The connectivity data consists of a list of data for each fill area set. The data for each fill area set consists of a 16-bit count of the number of contours defining the fill area set followed by a list of data for each contour. The data for each contour consists of a 16-bit count of the number of vertices defining the contour followed by the data for each vertex. Each vertex is defined as a 16-bit index into *per_vert_data*.

**Encoding: LISTofLISTofLISTofCARD16 for Set of Fill Area Sets with Data (Versions 5.0 and 5.2)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | number of contours in the first fill area set |
| 2 | CARD16 | number of vertices in the first contour of the first fill area set |
| 2 | CARD16 | index of the first vertex of the first contour of the first fill area set |
| ... | | |
| 2 | CARD16 | index of the last vertex of the first contour of the first fill area set |
| ... | | |
| 2 | CARD16 | number of vertices in the last contour of the first fill area set |
| 2 | CARD16 | index of the first vertex of the last contour of the first fill area set |
| ... | | |
| 2 | CARD16 | index of the last vertex of the last contour of the first fill area set |
| ... | | |
| 2 | CARD16 | number of contours in the last fill area set |
| 2 | CARD16 | number of vertices in the first contour of the last fill area set |
| 2 | CARD16 | index of the first vertex of the first contour of the last fill area set |
| ... | | |
| 2 | CARD16 | index of the last vertex of the first contour of the last fill area set |
| ... | | |
| 2 | CARD16 | number of vertices in the last contour of the last fill area set |
| 2 | CARD16 | index of the first vertex of the last contour of the last fill area set |
| ... | | |
| 2 | CARD16 | index of the last vertex of the last contour of the last fill area set |

### 6.7.33 Set of Fill Area Sets with Data (Version 5.2) 5.2

| | |
|---|---|
| *shape*: | SHAPE |
| *color_type*: | COLOR_TYPE |
| *fas_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *edge_attributes*: | BITMASK_SHORT |
| *contour_hint*: | CONTOUR |
| *contours_all_1*: | BOOLEAN |
| *num_fas*: | CARD16 |
| *num_verts*: | CARD16 |
| *num_edges*: | CARD16 |
| *num_contours*: | CARD16 |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *per_fas_data*: | LISTofOPT_DATA |
| *per_vert_data*: | LISTofVERTEX |
| *per_edge_data*: | LISTofSWITCH |
| *contours*: | LISTofLISTofLISTofCARD16 |

This primitive is identical to the Set of Fill Area Sets with Data (V5.0) primitive (§6.7.32) except this primitive allows facet and vertex floating-point data lists and 2D vertices to be optionally specified.

This command causes each fill area set in a set of fill area sets to be rendered independently. A set of fill area sets is a collection of not necessarily coplanar Fill Area Sets with Data (§6.7.13). The vertices are defined in a single list which is referenced by the definition of each fill area set. This allows vertex data to be defined once and then shared.

*num_verts* defines the number of vertices defined in the list of vertices, *per_vert_data*. The connectivity of these vertices is defined through *contours.* Each fill area set is defined by one or more lists of vertices. Each list forms one contour of the primitive. *num_contours* indicates the total number of contours in the primitive. *contours* indicates the number of contours in each fill area set, the number of vertices in each contour, and which vertices define each contour. The vertices are defined by referencing into *per_vert_data*, where the first vertex is referenced by index 0.

*contours_all_1* is *True* if there is only one contour in each fill area set. The results are implementation-dependent if *contours_all_1* is set to *True* when there are no contours in any or all of the fill area sets, or if there are more than one contour in any of the fill area sets.

*fas_attributes* indicates the optional data specified for each fill area set. The components of the *fas_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of these bits is 1, the corresponding data is available in the *num_fas* entries of *per_fas_data*. Otherwise *per_fas_data* is empty. Be aware if even a fill area set has zero contours defined, PEX still expects *per_fas_data* to contain an entry for that fill area set.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) 5.2 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of these bits is set, the corresponding data is indicated for each vertex.

If fill area set and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for fill area set and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the fill area set attributes and/or vertex attributes is set, then extended data is specified with each fill area set and/or vertex, respectively. If the *fp_data* fill area set attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each fill area set. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. Each field in *fp_data_counts* is ignored if its corresponding *fp_data* bit is not set.

If the 2D_coords vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

*edge_attributes* indicates the optional data specified for each edge. The components of the *edge_attributes* bitmask are:

| | |
|---|---|
| *edges*: | SWITCH |

Edge flags are present in *per_edge_data* if the *edges* bit is set. This edge data is used if the pipeline state’s *surface edge flag* is *On*. *per_edge_data* is a list of *num_edges* entries. The list corresponds to the order in which vertices are specified in *contours*, even if the vertices form a degenerate contour which is not drawn. The first edge flag in *per_edge_data* corresponds to the edge between the first and second vertices of the first contour of the first fill area set, and so on. Each entry of *per_edge_data* contains a switch indicating whether an edge is drawn from the corresponding vertex to the next vertex. The edge flag corresponding with the last vertex of each contour indicates whether an edge is drawn from the last vertex to the first vertex of the contour. If the *edges* bit is not set, the *per edge data* list is empty and all the edges are drawn if the pipeline state’s *surface edge flag* is *On*.

Usage of *shape* and *contour_hint* is as defined for the Fill Area Set primitive (§6.7.11).

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Set of Fill Area Sets with Data (V5.2).

An *OutputCommand* error is reported if any of the edge flags in the *per_edge_data* list is not *On* or *Off*, or if any of the *shape, contour_hint,* or *contours_all_1* parameters is invalid, or if any of the vertex indices in the *contours* array is out of range, or if there are not as many edge flags as they are vertex indices in the lists in the contours array.

**Encoding: Set of Fill Area Sets with Data (Version 5.2)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 148 | output command number |
| 2 | 0 OR 7+ (*fc*d+vc*v+ec*e+p1*+2**fc*+2**cc*+2**ec*+*p2*)/ 4 | output command length |
| $k^{\dagger 1}$ | 8+ (*fc*d+vc*v+ec*e+p1*+2**fc*+2**cc*+2**ec*+*p2*)/ 4 | 32-bit output command length |
| 2 | SHAPE | shape |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | fill area set attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | BITMASK_SHORT | edge attributes (call this value *e*) |
| | 0x0001 *individual_edges* | |
| 1 | CONTOUR | contour hint |
| 1 | BOOLEAN | all contour counts = 1 |
| 2 | CARD16 | number of fill area sets (*fc*) |
| 2 | CARD16 | number of vertices (*vc*) |
| 2 | CARD16 | number of edges (sum of all contour counts) (*ec*) |
| 2 | CARD16 | number of contours (*cc*) |
| 4 | FP_DATA_COUNTS | the number of floats in the fill area set and vertex floating-point data lists |
| *fc*d* | LISTofOPT_DATA | fill area set data (see OPT_DATA for *d*) |
| *vc*v* | LISTofVERTEX | vertex data (see VERTEX for *v*) |
| *ec*e* | LISTofSWITCH | edge data |
| *p1* | | pad(*ec*e*) |
| 2**fc*+2**cc*+ 2**ec* | LISTofLISTofLISTofCARD16 | connectivity data (see §6.7.32.1) |
| *p2* | | *p*=pad(2**fc*+2**cc*+2**ec*) |

†1) *k* = ((output command length == 0) : 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Set of Fill Area Sets with Data (V5.2).
†3) Only bits *color*, *normal*, *2D_coords*, and *fp_data* are supported by Set of Fill Area Sets with Data (V5.2).

### 6.7.34 Sphere Set with Data

| | |
|---|---|
| *radius*: | FLOAT |
| *color_type*: | COLOR_TYPE |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_count*: | FP_DATA_COUNTS |
| *spheres*: | LISTofVERTEX |

This command causes each sphere in a set of spheres to be rendered independently. Each sphere is a three-dimensional surface whose points are all equidistant from its center vertex. The center vertex and radius are both defined in modeling coordinates.

*vert_attributes* indicates the optional data specified for the center vertex of each sphere. The components of the bitmask are:

| | |
|---|---|
| *color:* | COLOR |
| *radius:* | FLOAT |
| *fp_data*: | LISTofFLOAT |

If any of these bits is 1, the corresponding data is indicated in *spheres*.

If the *color* bit is set, the color to be applied to each sphere is specified with its center vertex. The colors are of the type specified by *color_type*. If the *color* bit is not set, *color_type* is ignored.

If the *radius* bit is 1, the radius of each sphere is defined in *spheres* and parameter *radius* is ignored. Otherwise parameter *radius* defines the common radius applied to all the spheres in the set. If the radius of any sphere is less than or equal to zero, the results of rendering that sphere are implementation-dependent.

If the *fp_data* bit is set, then extended data is specified with each center vertex. The *vertex_fp_count* in *fp_data_count* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If the *fp_data* bit is not set, *fp_data_count* is ignored.

The General Surface, Styled and Shaded Surface, Curve, and Data mapping attribute sets and surface clipping are applied to Sphere Set with Data. When interior style *Hollow* is applied to the sphere set, nothing is rendered. The results when interior styles *Hatch* and *Pattern* are applied are implementation-dependent.

**Encoding: Sphere Set with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 111 | output command number |
| 2 | 4+(*fp*+*n***v*)/4 OR 0 | output command length |
| $k^{\dagger 1}$ | 5+(*fp*+*n***v*)/4 | 32-bit output command length |
| *fp* | FLOAT | radius |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | vertex attributes |
| 4 | FP_DATA_COUNTS | the number of floats in the vertex floating-point data list |
| 4 | CARD32 | number of spheres (*n*) |
| *n***v* | LISTofVERTEX | sphere data (see VERTEX for v) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *radius*, and *fp_data* are supported by Sphere Set with Data.

### 6.7.35 Text

5.0

*origin*: COORD_3D
*vector1*: VECTOR_3D
*vector2*: VECTOR_3D
*string*: LISTofMONO_ENCODING

This command causes a text string to be rendered. A text string is a set of characters which are located on a plane defined by *origin* and two direction vectors, *vector1* and *vector2*. *origin* is the position in modeling coordinates which defines the text local coordinate system origin. The text plane is the plane containing *origin* and which is perpendicular to the cross product $vector1 \times vector2$. The X-axis of text local coordinates is in the text plane and parallel to *vector1*. Let *v* be a vector perpendicular to *vector1* and in the text plane. *v* defines the positive Y-axis of text local coordinates if the dot product $v \cdot vector2$ is positive; otherwise the Y-axis is opposite *v*. Note the text local coordinate system has the same scale but a different orientation than the modeling coordinate system.

*string* is a list of mono-encodings. The rendered text string is formed from its mono-encodings in the order the mono-encodings are given. There is no space between the last character of a mono-encoding and the first character of the next mono-encoding beyond that indicated by the pipeline state's *character spacing* and the spacing incorporated in the character representations specific to the font used.

Each mono-encoding contains a *char_set*, *char_set_width,* and *encoding_state* field, as well as the list of characters that actually make up that portion of the string. The *char_set* is an index into the current font group. The current font group is the entry of the font table selected by the pipeline state's *text font index* attribute. A *char_set* value of one selects the first font in the font group. If a *char_set* value selects a member of the current font group which is not available, the entire string is rendered using the default font group. If a *char_set* value selects a member of the default font group which is not available, that portion of the string is rendered in an implementation-dependent manner. The *char_set_width* field of a mono-encoding indicates whether the characters are 8-bit, 16-bit, or 32-bit values. Characters are byte-swapped by the PEX server if necessary. The *encoding_state* field is provided for use by clients and is not interpreted by the server.

The Text and Annotation and Text attribute sets and text clipping are applied to Text.

**Encoding: Text**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 81 | output command number |
| 2 | 2+*s*+(9**fp*+*n*)/4 OR 0 | output command length |
| *k*†1 | 3+*s*+(9**fp*+*n*)/4 | 32-bit output command length |
| 3**fp* | COORD_3D | origin |
| 3**fp* | VECTOR_3D | vector 1 |
| 3**fp* | VECTOR_3D | vector 2 |
| 2 | CARD16 | number of mono-encodings (*s*) |
| 2 | | unused |
| 4**s*+*n*†2 | LISTofMONO_ENCODING | string |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) *n* is the sum of the sizes of the mono-encodings. See MONO_ENCODING (§3.7.7.84) for the mono-encoding size.

### 6.7.36 Text 2D

5.0

*origin*: COORD_2D

*string*: LISTofMONO_ENCODING

This command causes a text string to be rendered. The command is a 2D form of the Text output command (§6.7.35), and functions exactly as the 3D text primitive except is it always drawn in the X-Y, Z = 0 modeling coordinate plane, with the first direction vector aligned with the modeling coordinate *x*-axis and the second direction vector aligned with the modeling coordinate *y*-axis. The text string *origin* is a 2D value with the Z-coordinate implicitly zero.

**Encoding: Text 2D**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 82 | output command number |
| 2 | 2+*s*+(2**fp*+*n*)/4 OR 0 | output command length |
| *k*†1 | 3+*s*+(2**fp*+*n*)/4 | 32-bit output command length |
| 2**fp* | COORD_2D | origin |
| 2 | CARD16 | number of mono-encodings (*s*) |
| 2 | | unused |
| 4*s+*n*†2 | LISTofMONO_ENCODING | string |

†1) *k* = ((output command length == 0) ? 4 : 0)

†2) *n* is the sum of the sizes of the mono-encodings. See MONO_ENCODING (§3.7.7.84) for the mono-encoding size.

### 6.7.37 Triangle Fan with Data

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *facet_data*: | LISTofOPT_DATA |
| *vertices*: | LISTofVERTEX |

This command causes each triangle in a triangle fan to be rendered independently. A triangle fan is defined by a list of vertices in modeling coordinates. The first triangle is made up of vertices 1, 2 and 3, the second triangle is made up of vertices 1, 3, and 4, the third triangle of is made up of vertices 1, 4, and 5, and so on.

*facet_attributes* indicates the optional data specified for each triangle. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *fp_data*: | LISTofFLOAT |

If any of the attribute bits is set, the corresponding attributes are present in *facet_data*.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *edges*: | BITMASK |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) |
| *fp_data*: | LISTofFLOAT |

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the facet attributes and/or vertex attributes is set, then extended data is specified with each facet and/or vertex, respectively. If the *fp_data* facet attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each facet. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If only one *fp_data* bit is set, the size corresponding to the other *fp_data* bit is ignored. If neither *fp_data* bit is set, *fp_data_counts* is ignored.

If the *edges* bit of the vertex attributes is set, two surface edge flags are specified with each vertex. This edge data is used if the pipeline state's *surface edge flag* is *On*. For the *i*th vertex, the first edge flag bit (0x00000001) indicates whether to draw the edge between the *i*th and *i-1*th vertices, and the second flag bit (0x00000002) indicates whether to draw the edge between the *i*th and first vertices. The edge flags of the first vertex in *vertices* are ignored; the first edge flag for the second vertex is ignored. If the *edges* vertex attribute is not set, all edges are drawn if the pipeline state's *surface edge flag* is *On*.

If the *2D_coords* vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Triangle Fan with Data.

A triangle fan containing fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Triangle Fan with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 112 | output command number |
| 2 | 5+((*p*-2)**d*+*p***v*)/4 OR 0 | output command length |
| *k*†1 | 6+((*p*-2)**d*+*p***v*)/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of points (*p*) |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex floating-point data lists |
| (*p*-2)**d* | LISTofOPT_DATA | facet data (see OPT_DATA for *d*) |
| *p***v* | LISTofVERTEX | vertices (see VERTEX for *v*) |

†1) *k* = ((output command length == 0) ? 4 : 0)

†2) Only bits *color*, *normal*, and *fp_data* are supported for the facet attributes of Triangle Fan with Data.

†3) Only bits *color*, *normal, edges, 2D_coords*, and *fp_data* are supported for the vertex attributes of Triangle Fan with Data.

### 6.7.38 Triangle Strip with Data (Version 5.0) 5.0

*color_type*: COLOR_TYPE
*facet_attributes*: PRIM_DATA_ATTRIBUTES
*vert_attributes*: PRIM_DATA_ATTRIBUTES
*facet_data*: LISTofOPT_DATA
*vertices*: LISTofVERTEX

This primitive is identical to the Triangle Strip with Data (V5.2) primitive (§6.7.39) except this primitive does not allow facet floating-point data lists, vertex individual edge flags and floating-point data lists, and 2D vertices.

This command causes each triangle in a strip of triangles to be rendered independently. A triangle strip is defined by a list of vertices in modeling coordinates. The first triangle is made up of vertices 1, 2 and 3, the second triangle is made up of vertices 2, 3, and 4, and so on.

*facet_attributes* indicates the optional data specified for each triangle. The components of the *facet_attributes* bitmask are:

*color*: COLOR 5.0
*normal*: VECTOR_3D 5.0

If any of the attribute bits is set, the corresponding attributes are present in *facet_data*.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

*color*: COLOR 5.0
*normal*: VECTOR_3D 5.0

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Triangle Strip with Data (V5.0).

A triangle strip containing fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Triangle Strip with Data (Version 5.0)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 95 | output command number |
| 2 | $4+((p\text{-}2)*d + p*v)/4$ OR 0 | output command length |
| $k$†1 | $5+((p\text{-}2)*d + p*v)/4$ | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of points ($p$) |
| $(p\text{-}2)*d$ | LISTofOPT_DATA | facet data (see OPT_DATA for $d$) |
| $p*v$ | LISTofVERTEX | vertices (see VERTEX for $v$) |

†1) $k$ = ((output command length == 0) ? 4 : 0)
†2) Only bits *color* and *normal* are supported by Triangle Strip with Data (V5.0).
†3) Only bits *color* and *normal* are supported by Triangle Strip with Data (V5.0).

### 6.7.39 Triangle Strip with Data (Version 5.2) 5.2

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *facet_data*: | LISTofOPT_DATA |
| *vertices*: | LISTofVERTEX |

This primitive is identical to the Triangle Strip with Data (V5.0) primitive (§6.7.38) except this primitive allows facet floating-point data lists, vertex individual edge flags and floating-point data lists, and 2D vertices to be optionally specified.

This command causes each triangle in a strip of triangles to be rendered independently. A triangle strip is defined by a list of vertices in modeling coordinates. The first triangle is made up of vertices 1, 2 and 3, the second triangle is made up of vertices 2, 3, and 4, and so on.

*facet_attributes* indicates the optional data specified for each triangle. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of the attribute bits is set, the corresponding attributes are present in *facet_data*.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |
| *edges*: | BITMASK 5.2 |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified) 5.2 |
| *fp_data*: | LISTofFLOAT 5.2 |

If any of these bits is set, the corresponding data is indicated for each vertex.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *fp_data* bit of the facet attributes and/or vertex attributes is set, then extended data is specified with each facet and/or vertex, respectively. If the *fp_data* facet attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each facet. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. Each field in *fp_data_counts* is ignored if its corresponding *fp_data* bit is not set.

If the *edges* bit of the vertex attributes is set, two surface edge flags are specified with each vertex. This edge data is used if the pipeline state's *surface edge flag* is *On*. For the *i*th vertex, the first edge flag bit (0x00000001) of the vertex's edge bitmask indicates whether to draw the edge between the *i-1*th and *i*th vertex, and the second bit (0x00000002) of the edge bitmask indicates whether to draw the edge between the *i-2*th and *i*th vertex. The edge flags of the first vertex in *vertices* are ignored; the second edge flag for the second vertex is ignored. If the *edges* vertex attribute is not set, all the edges are drawn if the pipeline state's *surface edge flag* is *On*.

If the *2D_coords* vertex attribute bit is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Triangle Strip with Data (V5.2).

A triangle strip containing fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Triangle Strip with Data (Version 5.2)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 149 | output command number |
| 2 | 5+((*p*-2)**d* + *p***v*)/4 OR 0 | output command length |
| *k*†1 | 6+((*p*-2)**d* + *p***v*)/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of points (*p*) |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex floating-point data lists |
| (*p*-2)**d* | LISTofOPT_DATA | facet data (see OPT_DATA for *d*) |
| *p***v* | LISTofVERTEX | vertices (see VERTEX for *v*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Triangle Strip with Data (5.2).
†3) Only bits *color*, *normal*, *edges*, *2D_coords*, and *fp_data* are supported by Triangle Strip with Data (5.2).

### 6.7.40 Triangle Strip Fan Set with Data

| | |
|---|---|
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *num_facets*: | CARD32 |
| *num_verts*: | CARD32 |
| *fp_data_counts*: | FP_DATA_COUNTS |
| *facet_data*: | LISTofOPT_DATA |
| *vertices*: | LISTofVERTEX |

This command causes each triangle in a set of independent triangles and/or triangle fans and/or triangle strips to be rendered independently. A triangle strip fan set is defined by a list of vertices in modeling coordinates. The vertices form any combination of triangle subprimitives: triangle strips, triangle fans, and independent triangles.

*facet_attributes* indicates the optional data specified for each facet. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *fp_data*: | LISTofFLOAT |

If any of these bits is 1, the corresponding data is available in the *num_facets* entries of *facet_data*. Otherwise *facet_data* is empty.

*vert_attributes* indicates the optional data specified with each vertex. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR |
| *normal*: | VECTOR_3D |
| *edges*: | BITMASK |
| *2D_coords*: | (Indicates whether the Z-coordinate of the vertices is specified.) |
| *fp_data*: | LISTofFLOAT |

If any of these bits is set, the corresponding data is indicated for each vertex. The edge bitmask must be present for each vertex in the vertex data of a Triangle Strip Fan Set with Data since it includes the vertex control bits necessary to define this primitive. The *edges* bit of *vert_attributes* does not indicate the presence of the edge bitmask data as it does for all other primitives for which it is specified. For this primitive, the *edges* bit only indicates whether the edge flags bits of the edge bitmask are to be used if the pipeline state's *surface edge flag* is *On*.

If facet and/or vertex colors are defined, they are of the type specified by *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

If the *2D_coords* vertex attribute is set, the vertex coordinates are of type COORD_2D instead of type COORD_3D.

If the *fp_data* bit of the facet attributes and/or vertex attributes is set, then extended data is specified with each facet and/or vertex, respectively. If the *fp_data* facet attribute bit is set, then *facet_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each facet. If the *fp_data* vertex attribute bit is set, then *vertex_fp_count* in *fp_data_counts* indicates the number of floating-point values in the LISTofFLOAT for each vertex. If only one *fp_data* bit is set, the size corresponding to the other *fp_data* bit is ignored. If neither *fp_data* bit is set, *fp_data_counts* is ignored.

The edge bitmask must be defined for each vertex. Bits 0x00000004 and 0x00000008 of the bitmask are required since they are the control bits which define the subprimitives by defining the connectivity of the vertices. Table 6-19 shows how the control bits associated with each vertex define how the vertex is used in the definition of a triangle subprimitive.

**Table 6-19 . Interpretation of Vertex Control Bits**

| Bit 0x00000008 | Bit 0x00000004 | **Description** |
|---|---|---|
| 0 | 0 | The vertex is the first vertex of a triangle subprimitive. |
| 0 | 1 | The vertex is added and the oldest vertex of the three previous vertices used is dropped to form the next triangle in a triangle strip. |
| 1 | 0 | The vertex is added and the middle vertex of the three previous vertices used is dropped to form the next triangle in a triangle fan. |
| 1 | 1 | The vertex defines the second or third vertex in the first triangle of a triangle subprimitive. |

The control bits for the first and second vertices of the Triangle Strip Fan Set will be ignored. Instead it will be assumed they are 00 and 11 respectively. Therefore it is permissible for a Triangle Strip Fan Set which defines a **single** triangle subprimitive to set the control bits of all the vertices to 01 (for a triangle strip) or 10 (for a triangle fan).

If the *edges* bit of *vert_attributes* is set, edge flags are additionally specified in the required vertex edge bitmask. This edge data is used if the pipeline state’s *surface edge flag* is *On*. Table 6-19 indicates for each vertex $v_i$ (as defined by the control bits) which edges bit 0x00000002 and bit 0x00000001 control. If one of these bits is set, the corresponding edge is rendered. If the *edges* vertex attribute is not set, all the edges are drawn if the pipeline state’s *surface edge flag* is *On*.

**Table 6-20 . Interpretation of Edge Flag Bits**

| **Vertex** | **Edge Flag Bit** | **Corresponding Edge** |
|---|---|---|
| $v_0$ | 0x00000001 | None |
| | 0x00000002 | None |
| $v_1$ | 0x00000001 | Edge ($v_0$, $v_1$) |
| | 0x00000002 | None |
| $v_i$ †1 | 0x00000001 | Edge ($v_{i-1}$, $v_i$) |
| | 0x00000002 | Edge ($v_{i-2}$, $v_i$) |

†1) Where *i* is (2..*n*) inclusive, and *n* is the number of vertices defining a triangle subprimitive. $v_i$ represents the current vertex, $v_{i-1}$ the previous vertex, and $v_{i-2}$ the oldest vertex of the three vertices defining a subprimitive triangle.

The General Surface, Styled and Shaded Surface, Edge, and Data Mapping attribute sets and surface clipping are applied to Triangle Strip Fan Set with Data.

A Triangle Strip Fan Set containing one or more subprimitives with fewer than three vertices is stored in a structure when included in a **PEXStoreElements** request. The output command has no visual effect when rendered. Such a primitive has no visual effect when included in a **PEXRenderOutputCommands** request.

**Encoding: Triangle Strip Fan Set with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 114 | output command number |
| 2 | 6+(*nf**d+nv*v*)/4 OR 0 | output command length |
| *k*†1 | 7+(*nf**d+nv*v*)/4 | 32-bit output command length |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†2 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†3 | vertex attributes |
| 2 | | unused |
| 4 | CARD32 | number of facets (*nf*) |
| 4 | CARD32 | number of vertices (*nv*) |
| 4 | FP_DATA_COUNTS | the number of floats in the facet and vertex floating-point data lists |
| *nf*d* | LISTofOPT_DATA | facet data (see OPT_DATA for *d*) |
| *nv*v* | LISTofVERTEX†4 | vertices (see VERTEX for *v*) |

†1) *k* = ((output command length == 0) ? 4 : 0)
†2) Only bits *color*, *normal*, and *fp_data* are supported by Triangle Strip Fan Set.
†3) Only bits *color*, *normal*, *edge*, *2D_coords*, and *fp_data* are supported by Triangle Strip Fan Set.
†4) The edge flag bitmask must be defined with each vertex. Unlike for other primitives using type VERTEX, the *edges* bit of the vertex attributes bit does not indicate whether the edge flag bitmask is defined.

## 6.8 Primitive Output Commands for Versions 5.0 and 5.1 Compatibility

The primitive output commands in this section are superseded by version 5.2 primitive output commands. Table 6-21 shows the old output command and its replacement. To maintain backwards compatibility, support of the superseded primitive output commands is mandatory for implementations supporting major version 5 of the PEX protocol.

**Table 6-21 . Superseded Primitive Output Commands and Their Replacements**

| Superseded Primitive Output Command | Replacement Primitive Output Command |
|---|---|
| Fill Area with Data | Fill Area Set with Data (Version 5.0 or 5.2) |

### 6.8.1 Fill Area with Data 5.0

| | |
|---|---|
| *shape*: | SHAPE |
| *ignore_edges*: | BOOLEAN |
| *color_type*: | COLOR_TYPE |
| *facet_attributes*: | PRIM_DATA_ATTRIBUTES |
| *vert_attributes*: | PRIM_DATA_ATTRIBUTES |
| *facet*: | FACET |

This command causes a fill area primitive to be rendered. This primitive is identical to the Fill Area primitive (§6.7.9) except additional information can be specified for the fill area itself and for each vertex.

*facet_attributes* indicates the optional data specified for the fill area. The components of the *facet_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |

If any of the attribute bits is 1, the corresponding attributes are present in *facet*.

*vert_attributes* indicates the optional data specified for each vertex of the fill area. The components of the *vert_attributes* bitmask are:

| | |
|---|---|
| *color*: | COLOR 5.0 |
| *normal*: | VECTOR_3D 5.0 |

If any of the attribute bits is set, the corresponding attributes are present with each vertex.

If facet and/or vertex colors are defined, they are of the type specified by the primitive's parameter *color_type*. Otherwise *color_type* is ignored.

Unit vectors are assumed by PEX for facet and vertex normals. The visual effect of non-unit normal vectors is implementation-dependent.

Usage of *shape* and *ignore_edges* is as defined for the Fill Area primitive (§6.7.9).

The General Surface, Styled and Shaded Surface, and Edge attribute sets and surface clipping are applied to Fill Area with Data.

An *OutputCommand* error is reported if the *shape* or *ignore_edges* parameter is invalid.

**Encoding: Fill Area with Data**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 91 | output command number |
| 2 | 4+*f*/4 | output command length |
| 2 | SHAPE | shape |
| 1 | BOOLEAN | ignore edges |
| 1 | | unused |
| 2 | COLOR_TYPE | color type |
| 2 | PRIM_DATA_ATTRIBUTES†1 | facet attributes |
| 2 | PRIM_DATA_ATTRIBUTES†2 | vertex attributes |
| 2 | | unused |
| *f* | FACET | facet (see §6.8.1.1 for *f*) |

†1) Only bits *color* and *normal* are supported by Fill Area with Data.
†2) Only bits *color* and *normal* are supported by Fill Area with Data.

### 6.8.1.1 Type FACET

FACET : [facet_data : OPT_DATA,
vertices : LISTofVERTEX]

**Encoding: FACET**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *d* | OPT_DATA | facet data (see OPT_DATA for *d*) |
| 4 | CARD32 | number of vertices (*nv*) |
| *nv*v* | LISTofVERTEX | vertices (see VERTEX for *v*) |

## 6.9 Attribute Output Commands

This section gives the definitions of output commands that control an output primitive's attributes.

### 6.9.1 Add Names to Name Set

5.0

*names*: LISTofNAME

This command adds the names in *names* to the *name set* entry of the renderer's pipeline state. Functionalities such as highlighting, invisibility, and picking use filters of included and excluded names to determine whether the functionality should be applied to a primitive based on the pipeline state's *name set*.

Any name in *names* which is above the maximum supported value for a name is ignored. *MaxNamesetNames* can be inquired with request **PEXGetImpDepConstants**.

Add Names to Name Set is a member of the Global attribute set.

**Encoding: Add Names To Name Set OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 73 | output command number |
| 2 | 1+*n* | output command length |
| 4**n* | LISTofNAME | list of names (length implicit) |

### 6.9.2 Alpha Blend Function

*function*: ALPHA_BLEND_FUNC

This command sets the *alpha blend function* entry of the renderer's pipeline state. Alpha blending is enabled when the renderer's *transparency method* is *AlphaBlend*; otherwise the pipeline state's *alpha blend function* is ignored. The alpha blending function specifies how color coming out of the depth cueing stage and the previously rendered pixel are to be individually modulated. The results are summed to determine a color then sent through the color approximation stage. If an alpha buffer is supported (§ 10.1.3), the alpha component of the color is stored directly in the alpha buffer when the other color components are sent through the color approximation stage. Note the previously rendered pixels have already been through color approximation and must be transformed back into the rendering color model for alpha blending. If the color approximation or rendering color model are different from the values used for previously rendered pixels, this can result in incorrect color. When the color approximation and rendering color model are the same as the values used for previously rendered pixels, there still may be a loss of color accuracy due to infidelity in transforming back through the color approximation.

The registered values for the alpha blending functions are shown in Table 6-22 . If the value given in *function* is not supported by the server, function *NoBlend* is used.

**Table 6-22 . Enumerated Type *AlphaBlendingFunction***

| Value | Mnemonic | Description |
|---|---|---|
| | | C = Color without its alpha component<br>A = Alpha component of color<br>$_{src}$ = Color output from depth cueing<br>$_{out}$ = Color output from alpha blending<br>$_{dst}$ = Previously rendered color<br>In all cases, if $C_{out}$ or $A_{out}$ exceeds 1.0, the value is clamped to 1.0 |
| 0 | *NoBlend* | $C_{out} = C_{src}$<br>$A_{out} = A_{src}$ |
| 1 | *SimpleBlend* | $C_{out} = (A_{src} * C_{src}) + ((1\text{-}A_{src}) * C_{dst})$<br>$A_{out} = (A_{src} * A_{src}) + ((1\text{-}A_{src}) * A_{dst})$ |
| 2 | *Function2* | $C_{out} = 0$<br>$A_{out} = 0$ |
| 3 | *Function3* | $C_{out} = C_{src} * C_{dst}$<br>$A_{out} = A_{src} * A_{dst}$ |
| 4 | *Function4* | $C_{out} = C_{src} * (1 - C_{dst})$<br>$A_{out} = A_{src} * (1 - A_{dst})$ |
| 5 | *Function5* | $C_{out} = A_{src} * C_{src}$<br>$A_{out} = A_{src} * A_{src}$ |
| 6 | *Function6* | $C_{out} = (1 - A_{src}) * C_{src}$<br>$A_{out} = (1 - A_{src}) * A_{src}$ |
| 7 | *Function7* | $C_{out} = A_{dst} * C_{src}$<br>$A_{out} = A_{dst} * A_{src}$ |
| 8 | *Function8* | $C_{out} = (1 - A_{dst}) * C_{src}$<br>$A_{out} = (1 - A_{dst}) * A_{src}$ |

**Table 6-22 . Enumerated Type *AlphaBlendingFunction (Continued)***

| Value | Mnemonic | Description |
|---|---|---|
| 9 | *Function9* | $C_{out} = C_{dst}$<br>$A_{out} = A_{dst}$ |
| 10 | *Function10* | $C_{out} = C_{src} + C_{dst}$<br>$A_{out} = A_{src} + A_{dst}$ |
| 11 | *Function11* | $C_{out} = (C_{src} * C_{dst}) + C_{dst}$<br>$A_{out} = (A_{src} * A_{dst}) + A_{dst}$ |
| 12 | *Function12* | $C_{out} = (C_{src} * (1 - C_{dst})) + C_{dst}$<br>$A_{out} = (A_{src} * (1 - A_{dst})) + A_{dst}$ |
| 13 | *Function13* | $C_{out} = (A_{src} * C_{src}) + C_{dst}$<br>$A_{out} = (A_{src} * A_{src}) + A_{dst}$ |
| 14 | *Function14* | $C_{out} = ((1 - A_{src}) * C_{src}) + C_{dst}$<br>$A_{out} = ((1 - A_{src}) * A_{src}) + A_{dst}$ |
| 15 | *Function15* | $C_{out} = (A_{dst} * C_{src}) + C_{dst}$<br>$A_{out} = (A_{dst} * A_{src}) + A_{dst}$ |
| 16 | *Function16* | $C_{out} = ((1 - A_{dst}) * C_{src}) + C_{dst}$<br>$A_{out} = ((1 - A_{dst}) * A_{src}) + A_{dst}$ |
| 17 | *Function17* | $C_{out} = C_{src} + (C_{src} * C_{dst})$<br>$A_{out} = A_{src} + (A_{src} * A_{dst})$ |
| 18 | *Function18* | $C_{out} = (C_{src} * (1 - C_{dst})) + (C_{src} * C_{dst})$<br>$A_{out} = (A_{src} * (1 - A_{dst})) + (A_{src} * A_{dst})$ |
| 19 | *Function19* | $C_{out} = (1 - C_{src}) * C_{dst}$<br>$A_{out} = (1 - A_{src}) * A_{dst}$ |
| 20 | *Function20* | $C_{out} = C_{src} + ((1 - C_{src}) * C_{dst})$<br>$A_{out} = A_{src} + ((1 - A_{src}) * A_{dst})$ |
| 21 | *Function21* | $C_{out} = (C_{src} * C_{dst}) + ((1 - C_{src}) * C_{dst})$<br>$A_{out} = (A_{src} * A_{dst}) + ((1 - A_{src}) * A_{dst})$ |
| 22 | *Function22* | $C_{out} = A_{src} * C_{dst}$<br>$A_{out} = A_{src} * A_{dst}$ |
| 23 | *Function23* | $C_{out} = C_{src} + (A_{src} * C_{dst})$<br>$A_{out} = A_{src} + (A_{src} * A_{dst})$ |
| 24 | *Function24* | $C_{out} = ((1 - A_{src}) * C_{src}) + (A_{src} * C_{dst})$<br>$A_{out} = ((1 - A_{src}) * A_{src}) + (A_{src} * A_{dst})$ |
| 25 | *Function25* | $C_{out} = (A_{dst} * C_{src}) + (A_{src} * C_{dst})$<br>$A_{out} = 2 * (A_{src} * A_{dst})$ |
| 26 | *Function26* | $C_{out} = ((1 - A_{dst}) * C_{src}) + (A_{src} * C_{dst})$<br>$A_{out} = ((1 - A_{dst}) * A_{src}) + (A_{src} * A_{dst})$ |
| 27 | *Function27* | $C_{out} = (1 - A_{src}) * C_{dst}$<br>$A_{out} = (1 - A_{src}) * A_{dst}$ |

**Table 6-22 . Enumerated Type *AlphaBlendingFunction (Continued)***

| Value | Mnemonic | Description |
|---|---|---|
| 28 | *Function28* | $C_{out} = C_{src} + ((1 - A_{src}) * C_{dst})$<br>$A_{out} = A_{src} + ((1 - A_{src}) * A_{dst})$ |
| 29 | *Function29* | $C_{out} = (A_{dst} * C_{src}) + ((1 - A_{src}) * C_{dst})$<br>$A_{out} = A_{dst}$ |
| 30 | *Function30* | $C_{out} = ((1 - A_{dst}) * C_{src}) + ((1 - A_{src}) * C_{dst})$<br>$A_{out} = ((1 - A_{dst}) * A_{src}) + ((1 - A_{src}) * A_{dst})$ |
| 31 | *Function31* | $C_{out} = A_{dst} * C_{dst}$<br>$A_{out} = A_{dst} * A_{dst}$ |
| 32 | *Function32* | $C_{out} = C_{src} + (A_{dst} * C_{dst})$<br>$A_{out} = A_{src} + (A_{dst} * A_{dst})$ |
| 33 | *Function33* | $C_{out} = (A_{src} * C_{src}) + (A_{dst} * C_{dst})$<br>$A_{out} = (A_{src} * A_{src}) + (A_{dst} * A_{dst})$ |
| 34 | *Function34* | $C_{out} = ((1 - A_{src}) * C_{src}) + (A_{dst} * C_{dst})$<br>$A_{out} = ((1 - A_{src}) * A_{src}) + (A_{dst} * A_{dst})$ |
| 35 | *Function35* | $C_{out} = ((1 - A_{dst}) * C_{src}) + (A_{dst} * C_{dst})$<br>$A_{out} = ((1 - A_{dst}) * A_{src}) + (A_{dst} * A_{dst})$ |
| 36 | *Function36* | $C_{out} = (1 - A_{dst}) * C_{dst}$<br>$A_{out} = (1 - A_{dst}) * A_{dst}$ |
| 37 | *Function37* | $C_{out} = C_{src} + ((1 - A_{dst}) * C_{dst})$<br>$A_{out} = A_{src} + ((1 - A_{dst}) * A_{dst})$ |
| 38 | *Function38* | $C_{out} = (A_{src} * C_{src}) + ((1 - A_{dst}) * C_{dst})$<br>$A_{out} = (A_{src} * A_{src}) + ((1 - A_{dst}) * A_{dst})$ |
| 39 | *Function39* | $C_{out} = ((1 - A_{src}) * C_{src}) + ((1 - A_{dst}) * C_{dst})$<br>$A_{out} = ((1 - A_{src}) * A_{src}) + ((1 - A_{dst}) * A_{dst})$ |
| 40 | *Function40* | $C_{out} = (A_{dst} * C_{src}) + ((1 - A_{dst}) * C_{dst})$<br>$A_{out} = (A_{dst} * A_{src}) + ((1 - A_{dst}) * A_{dst})$ |
| 41 | *Function41* | $C_{out} = (\min(A_{src}, (1 - A_{dst})) * C_{src}) + C_{dst}$<br>$A_{out} = A_{src} + A_{dst}$ |

A pixel is transparent if its alpha value is 0.0; it is opaque if its alpha value is 1.0. If the color of a primitive to be alpha blended does not have an alpha component, the default alpha value is used. For surface primitives, the default alpha value is the *transmission coef* field of the pipeline state's reflection attributes subtracted from 1. For back-facing fill area primitives, the pipeline state's back-facing reflection attributes are similarly used. For all other primitives, the default alpha value is 1.0.

Alpha blending interacts with HLHSR, primitive antialiasing, drawing functions other than *GXcopy,* and plane masking other than all planes enabled in an implementation-dependent manner.

Alpha Blending Function is a member of the Global attribute set.

**Encoding: Alpha Blending Function OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 115 | output command number |
| 2 | 2 | output command length |
| 2 | ALPHA_BLEND_FUNCTION | alpha blending function |
| 2 | | unused |

### 6.9.3 Annotation Text Alignment 5.0

*alignment*: TEXT_ALIGNMENT

This command sets the *annotation text alignment* entry of the renderer's pipeline state. The annotation text alignment is used to position an annotation text primitive's extent relative to the text position (the origin of the primitive's text local coordinate system). The extent is an imaginary parallelogram whose sides are parallel to the X and Y axes of the annotation text's local coordinate system. The sides are bounded by the font-specific character bodies of the string's characters. The registered vertical text alignment values are listed in Table 6-45 and the registered horizontal text alignment values are listed in Table 6-46 (§6.9.89).

Annotation Text Alignment is a member of the Annotation attribute set.

An *OutputCommand* error is issued if the horizontal or vertical annotation text alignment is not a registered value.

**Encoding: Annotation Text Alignment OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 19 | output command number |
| 2 | 2 | output command length |
| 4 | TEXT_ALIGNMENT | horizontal and vertex annotation text alignment |

### 6.9.4 Annotation Text Height 5.0

*height*: FLOAT

This command sets the *annotation text height* entry of the renderer’s pipeline state with the magnitude of the value in *height*. The annotation text height defines the nominal height of an uppercase character from its baseline to its capline (see §6.9.89 for a definition of baseline and capline). It is specified in the annotation text primitive’s local coordinate system (see §6.7.1). If the specified height is not supported, the height is mapped to the nearest height supported.

Annotation Text Height is a member of the Annotation attribute set.

**Encoding: Annotation Text Height OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 16 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | height |

### 6.9.5 Annotation Text Path

5.0

*path*: TEXT_PATH

This command sets the *annotation text path* entry of the renderer's pipeline state. The annotation text path defines the direction along which annotation text characters are placed. The registered text paths are listed in Table 6-47 (§6.9.94).

Annotation Text Path is a member of the Annotation attribute set.

An *OutputCommand* error is issued if the value in *path* is not one of the registered path values.

**Encoding: Annotation Text Path OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 18 | output command number |
| 2 | 2 | output command length |
| 2 | TEXT_PATH | annotation text path |
| 2 | | unused |

### 6.9.6 Annotation Text Style 5.0

*style*: ATEXT_STYLE

This command sets the *annotation text style* entry of the renderer's pipeline state. The annotation text style describes if and how the annotation text offset is rendered. The registered annotation text styles are listed in Table 6-23 . If the value in *style* is not one of the registered annotation text styles, style *NotConnected* is used.

**Table 6-23 . Enumerated Type *ATextStyle***

| Value | Symbol | Description |
|---|---|---|
| 1 | *NotConnected* | The annotation text primitive has no rendered line connecting its annotation and reference points. 5.0 |
| 2 | *Connected* | The annotation text primitive includes a rendered line connecting its annotation and reference points. The line is rendered with line attributes. 5.0 |

Annotation Text Style is a member of the Annotation attribute set.

**Encoding: Annotation Text Style OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 20 | output command number |
| 2 | 2 | output command length |
| 2 | ATEXT_STYLE | annotation text style |
| 2 | | unused |

### 6.9.7 Annotation Text Up Vector

5.0

*up*: VECTOR_2D

This command sets the *annotation text up vector* entry of the renderer's pipeline state. The character up vector defines the up direction for characters. It is defined in text local coordinates (see §6.7.1).

If the annotation text up vector defined by *up* has a length of zero, vector <0, 1> is used.

Annotation Text Up Vector is a member of the Annotation attribute set.

**Encoding: Annotation Text Up Vector OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 17 | output command number |
| 2 | 1+*fp*/2 | output command length |
| 2**fp* | VECTOR_2D | annotation text up vector |

### 6.9.8 Backface Data Mapping Index

*index*: TABLE_INDEX

This command sets the *bf data map index* of the renderer's pipeline state. The backface data mapping index is used to select an entry of the renderer's *data mapping table* whenever the pipeline state's *bf data mapping method ASF* is *Bundled.*

Backface Data Mapping Index is a member of the Data Mapping attribute set.

An *OutputCommand* error is issued if the value in *index* is 0.

**Encoding: Backface Data Mapping Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 116 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | data mapping table index |
| 2 | | unused |

### 6.9.9 Backface Data Mapping Method

*data_mapping*: DATA_MAP_METHOD_AND_DATA

This command sets the *bf data map method* entry of the renderer's pipeline state. This entry defines the backface data mapping method if the pipeline state's *bf data map method ASF* is *Individual*. The backface data mapping method specifies the mapping of color or other application-specific data to intrinsic color. It is the first stage in the color pipeline. It applies only to the back-facing facets of fill area "with data" output primitives when the pipeline state's *facet distinguish flag* is *True*. Otherwise the pipeline state's *data map method* is used to data map back-facing facets. The registered data mapping methods are listed in Table 6-25 (§6.9.27). If the method given in *data_mapping* is not supported by the server, method *None* is used.

Backface Data Mapping Method is a member of the Data Mapping attribute set.

An *OutputCommand* error is issued if the data record size doesn't match the data mapping method or if an undefined source selector is given.

**Encoding: Backface Data Mapping Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 117 | output command number |
| 2 | 2+($n$+$p$)/4 | output command length |
| $d$ | DATA_MAP_METHOD_AND_DATA | data mapping method (see DATA_MAP_METHOD_AND_DATA for $d$) |

### 6.9.10 Backface Interior Bundle Index

*index*: TABLE_INDEX

This command sets the *bf interior bundle index* of the renderer’s pipeline state. The backface interior bundle index is used to select an entry of the renderer’s *interior bundle* whenever a bundlable back-facing interior attribute’s ASF is *Bundled* and the renderer’s *PHIGS interior bundle mode* is *True.* (If the renderer’s *PHIGS interior bundle mode* is *False*, the pipeline state’s *interior bundle index* is used instead and the *bf interior bundle index* is ignored.) The bundlable back-facing interior attributes are backface surface interior style, backface surface interior style index, backface surface color, and backface surface interpolation method.

Backface Interior Bundle Index is a member of the Styled and Shaded Surface attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Backface Interior Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 118 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | interior bundle index |
| 2 | | unused |

### 6.9.11 Backface Reflectance Bundle Index

*index*: TABLE_INDEX

This command sets the *bf reflectance bundle index* of the renderer's pipeline state. The backface reflectance bundle index is used to select an entry of the renderer's *interior bundle* whenever a bundlable back-facing reflectance attribute's ASF is *Bundled* and the renderer's *PHIGS interior bundle mode* is *True.* (If the renderer's *PHIGS interior bundle mode* is *False*, the pipeline state's *interior bundle index* is used instead and the *bf reflectance bundle index* is ignored.) The bundlable back-facing reflectance attributes are backface surface reflection model and backface surface reflection attributes.

Backface Reflectance Bundle Index is a member of the General Surface attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Backface Reflectance Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 119 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | interior bundle index |
| 2 | | unused |

### 6.9.12 Backface Surface Active Textures

*bf_active_textures*: LISTofTABLE_INDEX

This command replaces the values in the *bf active textures* entry of the renderer’s pipeline state. The ordered list of indices into the renderer’s texture mapping lookup table selects the textures that are sequentially applied, if the pipeline state’s *facet distinguish flag* is *True*, to back-facing surfaces whose interior style is *Texture*.

If the number of indices given exceeds the maximum number of simultaneously supported texture maps, those textures beyond the limit will be ignored. Inquire *MaxTextureMaps* with request **PEXGetImpDepConstants** to determine this limit.

Backface Surface Active Textures is a member of the Styled and Shaded Surface attribute set.

An *OutputCommand* error is issued if a texture mapping lookup table index of 0 is in the list.

**Encoding: Backface Surface Active Textures OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 143 | output command number |
| 2 | 2+(2**n*+*p*)/4 | output command length |
| 2 | CARD16 | number of backface active textures (*n*) |
| 2 | | unused |
| 2**n* | LISTofTABLE_INDEX | prioritized backface active textures |
| *p* | | *p* = pad(2**n*) |

### 6.9.13 Backface Surface Color

5.0

*color*: COLOR_SPECIFIER

This command sets the *bf surface color* entry of the renderer's pipeline state. This entry defines the backface surface color if the pipeline state's *bf surface color ASF* is *Individual*.

Output command Backface Surface Color Index can alternately be used to set a backface surface color of type *Indexed*.

If the color type given is not supported, a color of type *Indexed* with a value of 1 is used.

Backface Surface Color is a member of the Styled and Shaded Surface attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Backface Surface Color OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 39 | output command number |
| 2 | 2+*sc*/4 | output command length |
| 4+*sc* | COLOR_SPECIFIER | backface surface color (see COLOR for *sc*) |

### 6.9.14 Backface Surface Color Index

5.0

*index*: TABLE_INDEX

This command sets the *bf surface color* entry of the renderer's pipeline state. This command is used when the color is of type *Indexed*. The *bf surface color* entry defines the backface surface color if the pipeline state's *bf surface color ASF* is *Individual*.

Output command Backface Surface Color, which allows a color of any type to be specified, can alternately be used.

Backface Surface Color Index is a member of the Styled and Shaded Surfaced attributed set.

An *OutputCommand* error is issued if the index is 65535.

**Encoding: Backface Surface Color Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 38 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | indexed backface surface color |
| 2 | | unused |

### 6.9.15 Backface Surface Interior Style

5.0

*interior_style*: INTERIOR_STYLE

This command sets the *bf interior style* entry of the renderer's pipeline state. This entry defines the backface surface interior style if the pipeline state's *bf interior style ASF* is *Individual*. The backface interior style describes how the interior of backface surfaces is "filled" when the pipeline state's *facet distinguish flag* is *True*. Otherwise the pipeline state's *interior style* is used for backface surfaces. The registered values for the interior style are defined in Table 6-41 (§6.9.84).

Backface Surface Interior Style is a member of the Styled and Shaded Surface attribute set.

If the value given for the backface interior style is not one of the registered interior styles, an *OutputCommand* error is issued. If the value given for the backface interior style is not supported by the server, style *Hollow* is used.

**Encoding: Backface Surface Interior Style OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 36 | output command number |
| 2 | 2 | output command length |
| 2 | INTERIOR_STYLE | backface interior style |
| 2 | | unused |

### 6.9.16 Backface Surface Interior Style Index 5.0

*index*: TYPE_OR_TABLE_INDEX

This command sets the *bf interior style index* entry of the renderer’s pipeline state. This entry defines the backface interior style index if the pipeline state’s *bf interior style index ASF* is *Individual*. If the pipeline state’s *bf interior style* is *Pattern* or *Hatch*, the backface interior style index is used to further define how the interior of back-facing surfaces is “filled” if the pipeline state’s *facet distinguish flag* is *True.* Otherwise the pipeline state’s *interior style index* is applied.

For interior style *Hatch*, the backface style index determines the hatch style. The registered values for hatch styles are defined in Table 6-30 (§6.9.85). If the value given in *index* is not supported by the server, style *45Degrees* is used.

For interior style *Pattern*, the index selects an entry of the renderer’s *pattern table*. If the index given in *index* is not supported by the server, index one is used.

Backface Surface Interior Style Index is a member of the Styled and Shaded Surface attribute set.

**Encoding: Backface Surface Interior Style Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 37 | output command number |
| 2 | 2 | output command length |
| 2 | TYPE_OR_TABLE_INDEX | backface interior style index |
| 2 | | unused |

### 6.9.17 Backface Surface Interpolation Method

5.0

*surface_interp*: SURFACE_INTERP

This command sets the *bf surface interpolation method* entry of the renderer's pipeline state. This entry defines the backface surface interpolation method if the pipeline state's *bf surface interpolation method ASF* is *Individual*. The backface surface interpolation method specifies how color values are computed in the interior of back-facing surface primitives if the pipeline state's *facet distinguish flag* is *True*. Otherwise the pipeline state's *surface interp* is applied to the back-facing surfaces instead. The backface surface interpolation method applies to all components of color, including alpha. The registered values for the surface interpolation method are shown in Table 6-43 (§6.9.86). If the value given in *surface_interp* is not supported by the server, method *None* is used.

Backface Surface Interpolation Method is a member of the Styled and Shaded Surface attribute set.

**Encoding: Backface Surface Interpolation Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 42 | output command number |
| 2 | 2 | output command length |
| 2 | SURFACE_INTERP | backface interpolation method |
| 2 | | unused |

### 6.9.18 Backface Surface Reflection Attributes

5.0

*attr*: REFLECTION_ATTR

This command sets the *bf reflection attributes* entry of the renderer's pipeline state. This entry defines the backface reflection attributes if the pipeline state's *bf reflection attributes ASF* is *Individual*. The backface reflection attributes are used in the reflection calculation (§2.2.2.2) used to light back-facing surface primitives if the pipeline state's *facet distinguish flag* is *True*. Otherwise the pipeline state's *reflection attr* entry is applied to back-facing surfaces.

The *ambient coef, diffuse coef,* and *specular coef* entries of the backface reflection attributes define the fraction of ambient, diffuse, and specular light respectively reflected by a back-facing surface. A value of 0.0 means no light is reflected; 1.0 means 100% of the light is reflected.

The larger the value of the *specular conc* field of the backface reflection attributes, the shinier the back-facing surface.

Inquire *TransparencySupported* with request **PEXGetImpDepConstants** to determine whether the transmission coefficient is utilized in the reflection calculation. The *transmission coef* entry of the backface reflection attributes is used to define the default alpha component of back-facing surface primitives without an explicit alpha-component. This alpha value is determined by subtracting the transmission coefficient from 1, since a transmission coefficient of 0.0 means the surface is opaque, while a coefficient of 1.0 means the surface is transparent.

If the color type given is not supported, a backface specular color of type *Indexed* with a value of 1 is used.

Backface Surface Reflection Attributes is a member of the General Surface attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Backface Surface Reflection Attributes OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 40 | output command number |
| 2 | 2+(*sc*+5**fp*)/4 | output command length |
| 4+*sc*+5**fp* | REFLECTION_ATTR | backface reflection attributes (see COLOR for *sc*) |

### 6.9.19 Backface Surface Reflection Model

5.0

*reflection_model*: REFLECTION_MODEL

This command sets the *bf reflection model* entry of the renderer's pipeline state. This entry defines the backface reflection model if the pipeline state's *bf reflection model ASF* is *Individual*. The backface reflection model specifies which reflectance components are considered when the reflectance calculations are conducted on back-facing surfaces if the pipeline state's *facet distinguish flag* is *True*. Otherwise the pipeline state's *reflection model* entry is applied when lighting back-facing surfaces. The registered reflection model values are defined in Table 6-44 (§6.9.88). If the value in *reflection_model* is not supported by the server, *None* is used.

Backface Surface Reflection Model is a member of the General Surface attribute set.

**Encoding: Backface Surface Reflection Model OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 41 | output command number |
| 2 | 2 | output command length |
| 2 | REFLECTION_MODEL | backface reflection model |
| 2 | | unused |

### 6.9.20 Character Expansion

5.0

*expansion*: FLOAT

This command sets the *char expansion* entry of the renderer’s pipeline state. This entry defines the character expansion if the pipeline state’s *char expansion ASF* is *Individual*. The character width is determined from the pipeline state’s *char height* and *text font index* since each font includes a width : height ratio. This character width is multiplied by the magnitude of the character expansion factor. The closest width supported by the font to the resulting width is then used.

Character Expansion is a member of the Annotation and Text attribute set.

**Encoding: Character Expansion OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 8 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | character expansion factor |

### 6.9.21 Character Height

5.0

*height*: FLOAT

This command sets the *char height* entry of the renderer's pipeline state with the magnitude of the value in *height*. The text height defines the nominal height of an uppercase character from its baseline to its capline (see §6.9.89 for a definition of baseline and capline). The text height is defined in text local coordinates (see §6.7.35 and §6.7.1). If the specified height is not supported, the height is mapped to the nearest height supported.

Character Height is a member of the Text attribute set.

**Encoding: Character Height OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 12 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | character height (MCs) |

### 6.9.22 Character Spacing 5.0

*spacing*: FLOAT

This command sets the *char spacing* entry of the renderer’s pipeline state. This entry defines the character spacing if the pipeline state’s *char spacing ASF* is *Individual*. The character spacing, defined as a fraction of the character’s height, defines the spacing between adjacent characters. Specifying a negative value causes the characters to overlap.

Character Spacing is a member of the Annotation and Text attribute set.

**Encoding: Character Spacing OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 9 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | character spacing |

### 6.9.23 Character Up Vector 5.0

*up*: VECTOR_2D

This command sets the *char up vector* of the renderer's pipeline state. The character up vector, specified in text local coordinates (see §6.7.35 and §6.7.1), defines the up direction for characters in the text plane. If the vector defined by *up* has a length of zero, vector <0, 1> is used.

Character Up Vector is a member of the Text attribute set.

**Encoding: Text Up Vector OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 13 | output command number |
| 2 | 1+*fp*/2 | output command length |
| 2**fp* | VECTOR_2D | character up vector |

### 6.9.24 Color Approximation Index

5.0

*index*: TABLE_INDEX

This command sets the *color approx index* of the renderer's pipeline state. The color approximation index is used to select an entry of the renderer's *color approx table*. Color approximation defines how, at the end of the rendering pipeline, colors (except their alpha components) are transformed into pixel values.

Color Approximation is a member of the Global attribute set.

An *OutputCommand* error is issued if the value in *index* is 65535.

**Encoding: Color Approximation Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 70 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | color approximation table index |
| 2 | | unused |

### 6.9.25 Curve Approximation Method

*approx*: CURVE_APPROX

This command sets the *curve approx method* entry of the renderer's pipeline state. This entry defines the curve approximation if the pipeline state's *curve approx method ASF* is *Individual*. The curve approximation specifies the smoothing method to be used when rendering curved univariate primitives. The defined curve approximation methods and their usage of the *tolerance* parameter are defined in Table 6-24 .

**Table 6-24 . Enumerated Type *CurveApproxMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *(*implementation-dependent*)* | The curve is smoothed in an implementation-dependent manner. *tolerance* is ignored. |
| 2 | *ConstantBetweenKnots* | The curve is segmented with equal parametric increments between successive pairs of knots. If the curve is formed from the Arc Set with Data or Arc Area Set with Data primitive and therefore doesn't have explicit knots, the entire circle or ellipse is treated as having a single knot. If the primitive is a circular or elliptical arc, the circle or ellipse from which it is formed is considered to have a single knot.<br><br>The *tolerance* field controls the segmentation. If *tolerance* does not define an integer value, it is truncated and only the integer portion is used. If *tolerance* is less than or equal to zero, the curve will be evaluated only at the parameter limits and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, the curve will be evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by *tolerance* between each pair of knots. |
| 3 | *WCS_ChordalSize* | The curve is segmented such that the WC length of each segment (chord) is less than the *tolerance* (specified in WCs). |
| 4 | *NPC_ChordalSize* | The curve is segmented such that the NPC length of each chord is less than the *tolerance* (specified in NPCs). |
| 5 | *DC_ChordalSize* | The curve is segmented such that the DC length of each chord is less than the *tolerance* (specified in DCs). |
| 6 | *WCS_ChordalDev* | The curve is segmented such that the deviation in WCs between the curve and its segmented representation does not exceed the *tolerance* (specified in WCs). |
| 7 | *NPC_ChordalDev* | The curve is segmented such that the deviation in NPCs between the curve and its segmented representation does not exceed the *tolerance* (specified in NPCs). |
| 8 | *DC_ChordalDev* | The curve is segmented such that the deviation in DCs between the curve and its segmented representation does not exceed the *tolerance* (specified in DCs). |
| 9 | *WCS_Relative* | *tolerance* is a value in the range [0.0, 1.0] which indicates a relative quality of rendering to be maintained independent of scaling in WCs. A value of 1.0 indicates the highest possible quality. |

**Table 6-24 . Enumerated Type *CurveApproxMethod* (Continued)**

| Value | Mnemonic | Description |
|---|---|---|
| 10 | *NPC_Relative* | *tolerance* is a value in the range [0.0, 1.0] which indicates a relative quality of rendering to be maintained independent of scaling in NPCs. A value of 1.0 indicates the highest possible quality. 5.0 |
| 11 | *DC_Relative* | *tolerance* is a value in the range [0.0, 1.0] which indicates a relative quality of rendering to be maintained independent of scaling in DCs. A value of 1.0 indicates the highest possible quality. 5.0 |

Specification of lower relative approximation quality is intended to result in smoothing at relatively higher speeds. Likewise specification of relatively higher quality is intended to result in smoothing at possibly lower speeds. Implementations may smooth with a higher quality than requested.

If the curve approximation method is not one of the registered values, method 1 (implementation-dependent) is used. If the specified tolerance cannot be achieved, the most accurate value available for the method is used.

Curve Approximation Method is a member of the Curve attribute set.

**Encoding: Curve Approximation Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 26 | output command number |
| 2 | 2+*fp*/4 | output command length |
| 4+*fp* | CURVE_APPROX | curve approximation method and tolerance |

### 6.9.26 Data Mapping Index

*index*: TABLE_INDEX

This command sets the *data map index* of the renderer's pipeline state. The data mapping index is used to select an entry of the renderer's *data mapping table* whenever the pipeline state's *data mapping method ASF* is *Bundled.*

Data Mapping Index is a member of the Data Mapping attribute set.

An *OutputCommand* error is issued if the value in *index* is 0.

**Encoding: Data Mapping Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 120 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | data mapping table index |
| 2 | | unused |

### 6.9.27 Data Mapping Method

*data_mapping*: DATA_MAP_METHOD_AND_DATA

This command sets the *data map method* entry of the renderer’s pipeline state. This entry defines the data mapping method if the pipeline state’s *data map method ASF* is *Individual*. The data mapping method specifies the mapping of color or other application-specific data to intrinsic color. It is the first stage in the color pipeline.The registered data mapping methods are listed in Table 6-25 . If the method given in *data_mapping* is not supported by the server, method *None* is used.

**Table 6-25 . Enumerated Type *DataMappingMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *None* | The prioritized source of intrinsic color, from first to last, is vertex color (if defined with the primitive), facet color (if defined with the primitive), and surface color.<br>No data is defined for this method. |
| 1 | *Color* | The source of intrinsic color is the vertex color (if defined with the primitive), facet color (if defined with the primitive), or surface color.<br>A prioritized list, which includes at least one of these sources, is defined for this method. The list starts with the top choice. If there is no match between the prioritized list of source selectors and the data defined with a primitive being data mapped, or if the source selectors are otherwise invalid, the surface color is used as the source. |

Data Mapping Method is a member of the Data Mapping attribute set.

An *OutputCommand* error is issued if the data record size doesn’t match the data mapping method or if an undefined source selector is given.

**Encoding: Data Mapping Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 121 | output command number |
| 2 | 2+(*n*+*p*)/4 | output command length |
| *d* | DATA_MAP_METHOD_AND_DATA | data mapping method (see DATA_MAP_METHOD_AND_DATA for *d*) |

### 6.9.28 Depth Cue Index

5.0

*index*: TABLE_INDEX

This command sets the *depth cue index* of the renderer's pipeline state. The depth cue index is used to select an entry of the renderer's *depth cue table*. Depth cueing is a blending of the color computed by the lighting, shading, and texturing stages of the rendering pipeline with a depth cue color. The blending is controlled relative to the depth in NPC space. Often depth cueing is used to blend the portion of the image furthest away from the viewer with the background color, heightening the sensation of depth in the image.

Depth Cue Index is a member of the Global attribute set.

An *OutputCommand* error is issued if the value in *index* is 65535.

**Encoding: Depth Cue Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 67 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | depth cueing table index |
| 2 | | unused |

### 6.9.29 Drawing Function

*function*: DRAWING_FUNCTION

This command sets the *drawing function* entry of the renderer's pipeline state to the value given in *function*. The drawing function indicates how, after color approximation, pixels being newly drawn are combined with existing pixels in the drawable. It applies only to the image planes. It applies to the rendering of all output primitives. The registered drawing functions are listed in Table 6-26 .

**Table 6-26 . Drawing Function Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0 | *GXclear* | all 0's |
| 1 | *GXand* | src AND dest |
| 2 | *GXandReverse* | src AND (NOT dest) |
| 3 | *GXcopy* | src |
| 4 | *GXandInverted* | (NOT src) AND dest |
| 5 | *GXnoop* | dest |
| 6 | *GXxor* | src XOR dest |
| 7 | *GXor* | src OR dest |
| 8 | *GXnor* | (NOT src) AND (NOT dest) |
| 9 | *GXequiv* | (NOT src) XOR dest |
| 10 | *GXinvert* | NOT dest |
| 11 | *GXorReverse* | src OR (NOT dest) |
| 12 | *GXcopyInverted* | NOT src |
| 13 | *GXorInverted* | (NOT src) OR dest |
| 14 | *GXnand* | (NOT src) OR (NOT dest) |
| 15 | *GXset* | all 1's |

If echoing is enabled with method *XORColor* applied, the selected drawing function is overridden. Drawing functions other than *GXcopy* interact with alpha blending, primitive antialiasing, transparency, and plane masking other than all planes enabled in an implementation-dependent manner.

Drawing Function is a member of the Global attribute set.

An *OutputCommand* error is reported if *function* is not one of the registered drawing function values.

**Encoding: Drawing Function OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 122 | output command number |
| 2 | 2 | output command length |
| 2 | DRAWING_FUNCTION | drawing function |
| 2 | | unused |

### 6.9.30 Facet Culling Mode

5.0

*culling_mode*: CULL_MODE

This command sets the *facet culling mode* entry of the renderer's pipeline state. Enabling *BackFaces* or *FrontFaces* face culling causes back-facing facets or front-facing facets, respectively, to be thrown out and not rendered. A primitive facet is front-facing if its geometric normal has a non-negative Z-component in NPCs.

Facet Culling Mode is a member of the General Surface attribute set.

An *OutputCommand* error is reported if the mode is not *None*, *BackFaces*, or *FrontFaces*.

**Encoding: Facet Culling Mode OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 44 | output command number |
| 2 | 2 | output command length |
| 2 | CULL_MODE | facet culling mode |
| 2 | | unused |

### 6.9.31 Facet Distinguish Flag

5.0

*distinguish*: BOOLEAN

This command sets the *facet distinguish flag* entry of the renderer's pipeline state. When face distinguishing is *True,* the backface surface attributes are applied to back-facing facets, overriding the corresponding surface attributes. Otherwise the surface attributes are applied to both front-facing and back-facing facets. A primitive facet is front-facing if its geometric normal has a non-negative Z-component in NPCs.

Facet Distinguish Flag is a member of the General Surface attribute set.

An *OutputCommand* error is issued if the value in *distinguish* is not *True* or *False*.

**Encoding: Facet Distinguishing Flag OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 45 | output command number |
| 2 | 2 | output command length |
| 1 | BOOLEAN | facet distinguishing flag |
| 3 | | unused |

### 6.9.32 Generalized Structure Element

5.0

*gse_id*: INT32
*data*: data

This command allows implementation-specific pipeline state attributes to be set. Individual attributes are identified by their unique *gse_id*. For a particular type of GSE, the implementation must define the pipeline state attribute it sets, the contents of *data* and what values will result in an *OutputCommand* error being issued, and which primitives are affected.

A *gse_id* has its bit 32 (0x80000000) set, the vendor-specific identifier encoded in bits 17 through 31 (0x7FFF0000), and the GSE identifier in bits 1 through 16 (0x0000FFFF). The **PEXGetEnumeratedTypeInfo** request is used to query the GSEs supported by the server. However **PEXGetEnumeratedTypeInfo** can only return 16-bit enumerated type values, so it encodes a GSE value with bit 0x8000 set, the vendor-specific identifier is taken from bits 0x007F0000 of the 32-bit value and encoded in bits 0x7F00, and the GSE identifier is taken from bits 0x000000FF and encoded in bits 0x00FF. Where differing 32-bit GSE identifiers map to the same 16-bit *GSE* enumerated type value, the enumerated type mnemonics distinguish them. Byte-swapping and floating-point conversion are only done for GSE types supported by the server.

If a particular GSE type is not supported by the server, it is stored in a structure when included in a **PEXStoreElements** request but has no visual effect when rendered. The unsupported GSE also has no visual effect when included in a **PEXRenderOutputCommands** request.

Generalized Structure Element is a member of the Vendor-Specific attribute set.

**Encoding: Generalized Structure Element OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 78 | output command number |
| 2 | 3+(*n*+*p*)/4 | output command length |
| 4 | INT32 | GSE identifier |
| 2 | CARD16 | number of bytes of data (*n*) |
| 2 | | unused |
| *n* | | data |
| *p* | | *p*=pad(*n*) |

### 6.9.33 Global Transform 5.0

*matrix*: MATRIX

This command sets the *global transform* and *global transform hints* entries of the renderer’s pipeline state. This OC is identical to the Global Transform with Hints OC (§6.9.34) except the *global transform hints* cannot be explicitly set.

The server is required to update the pipeline state’s *global transform hints*. The server will likely set the hints to *none* since explicit matrix hints are not provided with this OC.

Global Transform is a member of the Global attribute set.

**Encoding: Global Transform OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 59 | output command number |
| 2 | 1+(16**fp*)/4 | output command length |
| 16**fp* | MATRIX | global transform matrix |

### 6.9.34 Global Transform with Hints

*matrix*: MATRIX

*matrix_hints*: MATRIX_HINTS

This command sets the *global transform* and *global transform hints* entries of the renderer's pipeline state. This OC is identical to the Global Transform OC (§6.9.33) except the *global transform hints* can be explicitly set with this OC.

Parameter *matrix_hints* is a bitmask used to provide the server knowledge about the characteristics of the matrix. The defined matrix hints are listed in Table 6-45 . Multiple matrix hints may be supplied by setting multiple bits in the bitmask. For example, if a matrix is formed by multiplying a translation matrix and a rotation matrix together, both the *translation* and *rotation* bits may be set. The server is required to update the pipeline state's *global transform hints*. Since the server is not required to use the hints, the server may set the *global transform hints* to *none*.

**Table 6-27 . Matrix Hints Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0x00000000 | *none* | There is no additional information about the contents of the matrix. |
| 0x00000001 | *identity* | The matrix contains: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 0x00000002 | *translation* | The matrix is non-singular, affine, angle-preserving, Euclidean, and one of the following forms, or a product of these forms. The fourth column in the fourth row is one: $\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 0x00000004 | *rotation* | The matrix is non-singular, affine, angle-preserving, Euclidean, and one of the following forms, or a product of these forms. The fourth column in the fourth row is one: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\Theta_x & -sin\Theta_x & 0 \\ 0 & sin\Theta_x & cos\ {}_{x\Theta} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\Theta_y & 0 & sin\Theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -sin\Theta_y & 0 & cos\Theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\Theta_z & -sin\Theta_z & 0 & 0 \\ sin\Theta_z & cos\Theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |

**Table 6-27 . Matrix Hints Constant Values (Continued)**

| **Value** | **Symbol** | **Description** |
|---|---|---|
| 0x00000008 | *reflection* | The matrix is non-singular, affine, angle-preserving, Euclidean transform with a negative determinant. The fourth column in the fourth row is one: $\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 0x00000010 | *uniform_scale* | The matrix is non-singular, affine, and angle-preserving. The fourth column in the fourth row is one: $\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 0x00000020 | *nonuniform_scale* | The matrix is non-singular, affine, and of one of the following forms, or a product of these forms. The fourth column in the fourth row is one: $\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |
| 0x00000040 | *shear* | The matrix is non-singular, affine, and of one of the following forms, or a product of these forms. The fourth column in the fourth row is one: $\begin{bmatrix} 1 & S_{xy} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{yx} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & S_{xz} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_{zx} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & S_{yz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & S_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ |

**Table 6-27 . Matrix Hints Constant Values (Continued)**

| Value | Symbol | Description |
|---|---|---|
| 0x00000080 | *perspective_projective* | The matrix is 3D to 3D projective, non-singular, non-affine, and of one of the following forms, or a product of these forms. All columns and rows have at least one non-zero value. One of the first three columns in the fourth row is non-zero $(r \neq 0)$ .The fourth column in the fourth row is one: $$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&r&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&r&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\r&0&0&1\end{bmatrix}$$ |
| 0x00000100 | *nonperspective_projective* | The matrix is 3D to 3D projective, non-singular, and non-affine. All columns and rows have at least one non-zero value. The fourth column in the fourth row is not equal to zero or one: $$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&s\end{bmatrix}$$ |
| 0x00000200 | *3D_to_2D_parallel_proj* | The matrix is singular and affine. One of the first three rows is all zeros. All of the first three columns in the fourth row are zero. The fourth column in the fourth row is one: $$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$ |
| 0x00000400 | *3D_to_2D_perspective_proj* | The matrix is singular and non-affine. One of the first three rows is all zeros. The corresponding column of the fourth row is non-zero $(r \neq 0)$ .The fourth column in the fourth row is one: $$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&r&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&r&0&1\end{bmatrix}\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\r&0&0&1\end{bmatrix}$$ |
| 0x00000800 | *zero* | The matrix is composed of all zero entries: $$\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$ |

**Table 6-27 . Matrix Hints Constant Values (Continued)**

| Value | Symbol | Description |
|---|---|---|
| 0x00001000 | *singular_and_unknown_type* | The matrix has a zero determinant. No other information is available. 5.2 |
| 0x00002000 | *nonsingular_and_unknown_type* | The matrix has a non-zero determinant. No other information is available. 5.2 |

Global Transform with Hints is a member of the Global attribute set.

The result of providing undefined hints or hints which do not accurately describe the matrix is implementation-dependent.

**Encoding: Global Transform with Hints OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 151 | output command number |
| 2 | 2+4**fp* | output command length |
| 16**fp* | MATRIX | global transform matrix |
| 4 | MATRIX_HINTS | matrix hints |

### 6.9.35 Global Transform 2D

5.0

*matrix*: MATRIX_3X3

This command sets the *global transform* and *global transform hints* entries of the renderer's pipeline state. Once *matrix* has been converted to a 4x4 matrix, this output command behaves exactly as the Global Transform output command (§6.9.33). The 3x3 *matrix* is transformed to a 4x4 matrix as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \Rightarrow \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{bmatrix} \qquad \textbf{Equation 6.5}$$

Global Transform 2D is a member of the Global attribute set.

**Encoding: Global Transform 2D OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 60 | output command number |
| 2 | 1+(9**fp*)/4 | output command length |
| 9**fp* | MATRIX_3X3 | matrix |

### 6.9.36 Global Transform 2D with Hints

*matrix*: MATRIX_3X3

*matrix_hints*: MATRIX_HINTS

This command sets the *global transform* and *global transform hints* entries of the renderer’s pipeline state. Once *matrix* has been converted to a 4x4 matrix, this output command behaves exactly as the Global Transform with Hints output command (§6.9.34). The 3x3 *matrix* is transformed to a 4x4 matrix as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \Rightarrow \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{bmatrix}$$

**Equation 6.6**

The matrix hints are representative of the expanded (4x4) matrix.

Global Transform 2D with Hints is a member of the Global attribute set.

**Encoding: Global Transform 2D with Hints OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 152 | output command number |
| 2 | 2+9**fp*/4 | output command length |
| 9**fp* | MATRIX_3X3 | matrix |
| 4 | MATRIX_HINTS | matrix hints |

### 6.9.37 Highlight Index

*index*: TABLE_INDEX

This command sets the *highlight index* of the renderer's pipeline state. The highlight index is used to select an entry of the renderer's *highlight table*. When the pipeline state's *name set* and the highlight filter defined by the renderer's *highlight inclusion* and *highlight exclusion* entries indicate a primitive is to be highlighted, the selected entry of the highlight table defines the highlight method.

Highlight Index is a member of the Global attribute set.

An *OutputCommand* error is issued if the value in *index* is 0.

**Encoding: Highlight Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 124 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | highlight table index |
| 2 | | unused |

### 6.9.38 HLHSR Identifier

*id*: HLHSR_IDENTIFIER

This command sets the *HLHSR identifier* entry of the renderer's pipeline state. The HLHSR identifier is not used with any registered HLHSR modes (§10.2.20) except *ZBufferId*. Table 6-28 shows the control over HLHSR mode *ZBufferId* provided by the HLHSR identifier. The effect of the HLHSR identifier on non-registered HLHSR modes is implementation-dependent. If the value given in *id* is not supported by the server, the HLHSR identifier used is implementation-dependent.

HLHSR identifiers are encoded in the least-significant 16 bits (0x0000FFFF). The most-significant 16 bits (0xFFFF0000) are used to encode whether the identifier is a registered identifier or a vendor extension. Request **PEXGetEnumeratedTypeInfo**, which is used to query the HLHSR identifiers supported by the server, can return only 16-bit enumerated type values. So **PEXGetEnumeratedTypeInfo** will encode the identifier in the least-significant 8 bits (0x00FF) and use the most-significant 8 bits (0xFF00) to encode whether the identifier is registered or a vendor extension. So it is possible for differing HLHSR identifiers to have identical values when queried. In this case, the mnemonics returned by **PEXGetEnumeratedTypeInfo** will distinguish the differing identifiers.

**Table 6-28 . Enumerated Type *HLHSRIdentifier***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *Disable* | Z-buffering is disabled. |
| 1 | *Enable* | Each pixel's Z-value is tested against the Z-buffer to determine if the pixel should be rendered. If the pixel is rendered, the Z-buffer is then updated to reflect the pixel's Z-value. |
| 2 | *EnableTestOnly* | Each pixel's Z-value is tested against the Z-buffer to determine if the pixel should be rendered. The Z-buffer is **not** updated to reflect the pixel's Z-value. |

HLHSR interacts with alpha blending, primitive antialiasing, transparency, drawing functions other than *GXcopy*, and plane masking other than all planes enabled in an implementation-dependent manner.

HLHSR Identifier is a member of the Global attribute set.

**Encoding: HLHSR Identifier OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 69 | output command number |
| 2 | 2 | output command length |
| 4 | HLHSR_IDENTIFIER | HLHSR identifier |

### 6.9.39 Individual ASF 5.0

*attribute*: ASF_ATTRIBUTE

*source*: ASF_VALUE

This command sets the indicated aspect source flag (ASF) entry of the renderer's pipeline state. The ASF is used to determine the source of the corresponding attribute's value. If the source is *Individual*, the attribute's value is taken directly from the pipeline state. If the source is *Bundled*, the attribute's value is taken from the corresponding bundle table. The entry of the bundle table used is determined from the bundle table's index in the pipeline state.

Individual ASF is a member of the Global attribute set.

An *OutputCommand* error is issued if *attribute* does not refer to a registered ASF, if multiple ASFs are selected, or if *source* is not set to *Bundled* or *Individual*.

**Encoding: Set Individual ASF OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 56 | output command number |
| 2 | 3 | output command length |
| 4 | ASF_ATTRIBUTE | ASF attribute |
| | 0x00000001 *marker_type_ASF* 5.0 | |
| | 0x00000002 *marker_scale_ASF* 5.0 | |
| | 0x00000004 *marker_color_ASF* 5.0 | |
| | 0x00000008 *text_font_index_ASF* 5.0 | |
| | 0x00000010 *text_prec_ASF* 5.0 | |
| | 0x00000020 *char_expansion_ASF* 5.0 | |
| | 0x00000040 *char_spacing_ASF* 5.0 | |
| | 0x00000080 *text_color_ASF* 5.0 | |
| | 0x00000100 *line_type_ASF* 5.0 | |
| | 0x00000200 *line_width_ASF* 5.0 | |
| | 0x00000400 *line_color_ASF* 5.0 | |
| | 0x00000800 *curve_approx_ASF* 5.0 | |
| | 0x00001000 *polyline_interp_ASF* 5.0 | |
| | 0x00002000 *interior_style_ASF* 5.0 | |
| | 0x00004000 *interior_style_index_ASF* 5.0 | |
| | 0x00008000 *surface_color_ASF* 5.0 | |
| | 0x00010000 *surface_interp_ASF* 5.0 | |
| | 0x00020000 *reflection_model_ASF* 5.0 | |
| | 0x00040000 *reflection_attr_ASF* 5.0 | |
| | 0x00080000 *bf_interior_style_ASF* 5.0 | |
| | 0x00100000 *bf_interior_style_index_ASF* 5.0 | |
| | 0x00200000 *bf_surface_color_ASF* 5.0 | |
| | 0x00400000 *bf_surface_interp_ASF* 5.0 | |
| | 0x00800000 *bf_reflection_model_ASF* 5.0 | |
| | 0x01000000 *bf_reflection_attr_ASF* 5.0 | |
| | 0x02000000 *surface_approx_ASF* 5.0 | |
| | 0x04000000 *surface_edges_ASF* 5.0 | |
| | 0x08000000 *surface_edge_type_ASF* 5.0 | |

**Encoding: Set Individual ASF OC (Continued)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| | 0x10000000 *surface_edge_width_ASF* 5.0 | |
| | 0x20000000 *surface_edge_color_ASF* 5.0 | |
| | 0x40000000 *parametric_surface_characteristics_ASF* 5.2 | |
| | 0x80000001 *data_map_method_ASF* 5.2 | |
| | 0x80000002 *bf_data_map_method_ASF* 5.2 | |
| | 0x80000004 *line_cap_style_ASF* 5.2 | |
| | 0x80000008 *line_join_style_ASF* 5.2 | |
| 1 | ASF_VALUE | ASF value |
| | 0 *Bundled* 5.0 | |
| | 1 *Individual* 5.0 | |
| 3 | | unused |

### 6.9.40 Interior Bundle Index

5.0

*index*: TABLE_INDEX

This command sets the *interior bundle index* of the renderer's pipeline state. Whenever the renderer's *PHIGS interior bundle mode* attribute is *False*, the interior bundle index is used to select an entry of the renderer's *interior bundle* to resolve any bundlable surface attribute whose ASF is *Bundled.*

If the renderer's *PHIGS interior bundle mode* is *True*, the interior bundle index still selects an entry of the renderer's *interior bundle* but only to resolve the bundlable interior attributes (surface interior style, surface interior style index, surface color, and surface interpolation method) whose ASFs are *Bundled*. (The pipeline state's *bf interior bundle index*, *reflectance bundle index, bf reflectance bundle index*, and *parametric surface bundle index* entries are used to resolve the other bundled surface attributes when the *PHIGS interior bundle mode* is *True*.)

Interior Bundle Index is a member of the General Surface, Styled and Shaded Surface, Curved Surface, and Parametric Surface attribute sets.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Interior Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 49 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | interior bundle index |
| 2 | | unused |

### 6.9.41 Light Source State

5.0

*enable*: LISTofTABLE_INDEX
*disable*: LISTofTABLE_INDEX

This command modifies the *light source state* entry of the renderer’s pipeline state. If the pipeline state’s *reflection model* or *bf reflection model* is any value other than *NoShading*, lighting is enabled. The light source state defines which lights are on by maintaining a list of indices into the renderer’s *light table*, each of which defines a light.

The list of light table indices in *enable* are added to the light source state. The list of light table indices in *disable* are removed from the light source state.

When processed by a renderer, this command will modify the pipeline state’s *light state* attribute. The current *light state* is modified by activating (“turning on”) each light source whose index is specified in the *enable* list, and by deactivating (“turning off”) each light source whose index is specified in the *disable* list. If any light in the *enable* or *disable* list references an undefined *Light* table entry, the light is ignored. If there are more non-ambient lights enabled than the server can consider in its reflectance calculations, it is implementation-dependent which lights are ignored. *MaxNonAmbientLights* is inquired with request **PEXGetImpDepConstants**.

Light Source State is a member of the General Surface attribute set.

An *OutputCommand* error is issued if a light table index of 0 is in either list, or if the same light is specified in both lists.

**Encoding: Light Source State OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 66 | output command number |
| 2 | 2+(2*$e$+2*$d$+$p1$+$p2$)/4 | output command length |
| 2 | CARD16 | number of lights to enable ($e$) |
| 2 | CARD16 | number of lights to disable ($d$) |
| 2*$e$ | LISTofTABLE_INDEX | light table indices of lights to enable |
| $p1$ | | $p1$=pad(2*$e$) |
| 2*$d$ | LISTofTABLE_INDEX | light table indices of lights to disable |
| $p2$ | | $p2$=pad(2*$d$) |

### 6.9.42 Line Bundle Index

5.0

*index*: TABLE_INDEX

This command sets the *line bundle index* of the renderer's pipeline state. The line bundle index is used to select an entry of the renderer's *line bundle* whenever a bundlable line attribute's ASF is *Bundled.*

Line Bundle Index is a member of the Line, Shaded Line, and Curve attribute sets.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Line Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 28 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | line bundle table index |
| 2 | | unused |

### 6.9.43 Line Cap Style

*line_cap_style*: LINE_CAP_STYLE

This command sets the *line cap style* entry of the renderer’s pipeline state. This entry defines the line cap style if the pipeline state’s *line cap style ASF* is *Individual*. The line cap style provides control over how the ends of a wide line are rendered. The registered values for line cap style are shown in Table 6-29 . If the value given in *line_cap_style* is not supported by the server, style *Butt* is used.

**Table 6-29 . Enumerated Type *LineCapStyle***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *Butt* | The wide line is square at the endpoint (perpendicular to the slope of the line) with no projection beyond. |
| 1 | *Round* | The wide line has a circular arc with the diameter equal to the linewidth, centered on the endpoint. |
| 2 | *Project* | The wide line is square at the end, but the path continues beyond the endpoint for a distance equal to half the linewidth. |

Line Cap Style is a member of the Line attribute set.

**Encoding: Line Cap Style OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 125 | output command number |
| 2 | 2 | output command length |
| 2 | LINE_CAP_STYLE | line cap style |
| 2 | | unused |

### 6.9.44 Line Color

5.0

*color*: COLOR_SPECIFIER

This command sets the *line color* entry of the renderer's pipeline state. This entry defines the line color if the pipeline state's *line color ASF* is *Individual*.

Output command Line Color Index can alternately be used to set a line color of type *Indexed*.

If the color type given is not supported, a color of type *Indexed* with a value of 1 is used.

Line Color is a member of the Line attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Line Color OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 25 | output command number |
| 2 | 2+*sc*/4 | output command length |
| 4+*sc* | COLOR_SPECIFIER | line color (see COLOR for *sc*) |

### 6.9.45 Line Color Index

5.0

*index*: TABLE_INDEX

This command sets the *line color* entry of the renderer's pipeline state. This command is used when the color is of type *Indexed*. The *line color* entry defines the line color if the pipeline state's *line color ASF* is *Individual*.

Output command Line Color, which allows a color of any type to be specified, can alternately be used.

Line Color Index is a member of the Line attribute set.

An *OutputCommand* error is issued if the index is 65535.

**Encoding: Line Color Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 24 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | indexed line color |
| 2 | | unused |

### 6.9.46 Line Join Style

*line_join_style*: LINE_JOIN_STYLE

This command sets the *line join style* entry of the renderer's pipeline state. This entry defines the line join style if the pipeline state's *line join style ASF* is *Individual*. The line join style provides control over how the intersection between adjacent segments of a wide line are rendered. The registered values for line join style are shown in Table 6-30 . If the value given in *line_join_style* is not supported by the server, style 0 (implementation-dependent) is used.

**Table 6-30 . Enumerated Type *LineJoinStyle***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *(implementation-dependent)* | The wide line segments are joined in an implementation-dependent fashion. |
| 1 | *Round* | Two wide line segments are joined by rendering a circular arc whose diameter is equal to the linewidth and whose centerpoint is the vertex between the two segments. |
| 2 | *Miter* | The out edges of the wide line segments extend to meet at an angle. However if the line miter limit (§6.9.47) is exceeded, style *Miter* is rendered as style *Bevel*. |
| 3 | *Bevel* | The two wide line segments are joined by applying cap style *Butt* to each of them and rendering the triangular notch left between them. |

Line Join Style is a member of the Line attribute set.

**Encoding: Line Join Style OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 126 | output command number |
| 2 | 2 | output command length |
| 2 | LINE_JOIN_STYLE | line join style |
| 2 | | unused |

### 6.9.47 Line Miter Limit

*limit*: FLOAT

This command sets the *line miter limit* entry of the renderer's pipeline state. The line miter limit determines the treatment of wide line segments joined by line join style *Miter*. If the ratio of the miter length, the distance between the inside and outside intersections of the two segments joined, to the line width exceeds the line miter limit, the line join style is interpreted as style *Bevel*.

Line Miter Limit is a member of the Line attribute set.

An OutputCommand error is issued if limit is less than 1.0.

**Encoding: Line Miter Limit OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 127 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | line miter limit |

### 6.9.48 Line Type 5.0

*line_type*: LINE_TYPE

This command sets the *line type* entry of the renderer's pipeline state. This entry defines the line type if the pipeline state's *line type ASF* is *Individual*. The line type describes the style used when rendering line primitives. The registered values for line type are defined in Table 6-31 . If the value given for line type is not supported by the server, type *Solid* is used.

**Table 6-31 . Enumerated Type *LineType***

| Value | Mnemonic | Description†1 |
|---|---|---|
| 1 | *Solid* | Draw the polyline or curve with a solid, unbroken line. 5.0 |
| 2 | *Dashed* | Draw the polyline or curve with a line that is dashed. 5.0 |
| 3 | *Dotted* | Draw the polyline or curve with a line that is dotted. 5.0 |
| 4 | *DashDot* | Draw the polyline or curve with a line that contains alternating dots and dashes. 5.0 |
| 5 | *CenterLine* | Draw the polyline or curve with a line that contains alternating short dashes and long dashes. 5.2 |
| 6 | *Phantom* | Draw the polyline or curve with a line that contains a sequence of a long dash followed by two short dashes. 5.2 |

†1) It is implementation-dependent whether the sequence for the *Dashed, Dotted, DashDot, CenterLine,* and *Phantom* line types is restarted or continued at the start of the polyline, at the start of a clipped segment of a polyline, and at each vertex of a polyline.

Line Type is a member of the Line attribute set.

**Encoding: Line Type OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 22 | output command number |
| 2 | 2 | output command length |
| 2 | LINE_TYPE | line type |
| 2 | | unused |

### 6.9.49 Line Width

5.0

*width*: FLOAT

This command sets the *line width* entry of the renderer’s pipeline state. This entry defines the line width if the pipeline state’s *line width ASF* is *Individual*. When a line primitive is rendered, the nominal line width is multiplied by the line width. The result is mapped to the nearest line width the server can render, so a negative line width results in the thinnest available line. The *NominalLineWidth*, *MinLineWidth*, *MaxLineWidth,* and *NumSupportedLineWidths* can be inquired with request **PEXGetImpDepConstants**.

Line Width is a member of the Line attribute set.

**Encoding: Line Width OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 23 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | line width scale factor |

### 6.9.50 Local Transform

5.0

*comp_type*: MODEL_MATRIX_COMPOSITION

*matrix*: MATRIX

This command sets the *local transform* and *local transform hints* entries of the renderer's pipeline state. This OC is identical to Local Transform with Hints (§6.9.51) except hints about *matrix* cannot be explicitly given with this OC. The pipeline state's *local transform* is modified by pre-concatenating it with *matrix*, post-concatenating it with *matrix*, or replacing it with *matrix*, as specified by the value in *comp_type*.

The server is required to update the pipeline state's *local transform hints* with a reasonable value after updating the *local transform.* The server may set the *local transform hints* to *none*, which it will likely do since explicit matrix hints are not provided with this primitive.

An *OutputCommand* error is reported if *comp_type* is not *PreConcatenate*, *PostConcatenate* or *Replace*.

Local Transform is a member of the Global attribute set.

**Encoding: Local Transform OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 57 | output command number |
| 2 | 2+(16**fp*)/4 | output command length |
| 2 | MODEL_MATRIX_COMPOSITION | composition type |
| 2 | | unused |
| 16**fp* | MATRIX | matrix |

### 6.9.51 Local Transform with Hints

| | |
|---|---|
| *comp_type*: | MODEL_MATRIX_COMPOSITION |
| *matrix*: | MATRIX |
| *matrix_hints*: | MATRIX_HINTS |

This command sets the *local transform* and *local transform hints* entries of the renderer's pipeline state. This OC is identical to Local Transform (§6.9.50) except hints about *matrix* can be explicitly given. The pipeline state's *local transform* is modified by pre-concatenating it with *matrix*, post-concatenating it with *matrix*, or replacing it with *matrix*, as specified by the value in *comp_type*.

Parameter *matrix_hints* is a bitmask used to provide the server knowledge about the characteristics of *matrix*. The defined matrix hints are listed in §6.9.34. Multiple matrix hints may be supplied by setting multiple bits in the bitmask. For example, if the matrix is formed by multiplying a translation matrix and a rotation matrix together, both the *translation* and *rotation* bits may be set. The server is required to update the pipeline state's *local transform hints* with a reasonable value after updating the *local transform.* Since the server is not required to use the hints, the server may set *local transform hints* to *none.*

The result of providing undefined hints or hints which do not accurately describe *matrix* is implementation-dependent. An *OutputCommand* error is reported if *comp_type* is not *PreConcatenate*, *PostConcatenate* or *Replace*.

Local Transform with Hints is a member of the Global attribute set.

**Encoding: Local Transform with Hints OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 153 | output command number |
| 2 | 3+4*fp* | output command length |
| 2 | MODEL_MATRIX_COMPOSITION | composition type |
| 2 | | unused |
| 16*fp* | MATRIX | matrix |
| 4 | MATRIX_HINTS | matrix hints |

### 6.9.52 Local Transform 2D

5.0

*comp_type*: MODEL_MATRIX_COMPOSITION

*matrix*: MATRIX_3X3

This command sets the *local transform* and *local transform hints* entries of the renderer's pipeline state. Once *matrix* has been converted to a 4x4 matrix, this output command behaves exactly as the Local Transform output command (§6.9.50). The 3x3 *matrix* is transformed to a 4x4 matrix as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \Rightarrow \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{bmatrix}$$

**Equation 6.7**

Local Transform 2D is a member of the Global attribute set.

**Encoding: Local Transform 2D OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 58 | output command number |
| 2 | 2+(9**fp*)/4 | output command length |
| 2 | MODEL_MATRIX_COMPOSITION | composition type |
| 2 | | unused |
| 9**fp* | MATRIX_3X3 | matrix |

### 6.9.53 Local Transform 2D with Hints

| | |
|---|---|
| *comp_type*: | MODEL_MATRIX_COMPOSITION |
| *matrix*: | MATRIX_3X3 |
| *matrix_hints*: | MATRIX_HINTS |

This command sets the *local transform* and *local transform hints* entries of the renderer’s pipeline state. Once *matrix* has been converted to a 4x4 matrix, this output command behaves exactly as the Local Transform with Hints output command (§6.9.51). The 3x3 *matrix* is transformed to a 4x4 matrix as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \Rightarrow \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{bmatrix}$$

**Equation 6.8**

The *matrix_hints* are representative of *matrix* after it has been expanded to a (4x4) matrix.

Local Transform 2D with Hints is a member of the Global attribute set.

**Encoding: Local Transform 2D with Hints OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 154 | output command number |
| 2 | 3+(9**fp*)/4 | output command length |
| 2 | MODEL_MATRIX_COMPOSITION | composition type |
| 2 | | unused |
| 9**fp* | MATRIX_3X3 | matrix |
| 4 | MATRIX_HINTS | matrix hints |

### 6.9.54 Marker Bundle Index

5.0

*index*: TABLE_INDEX

This command sets the *marker bundle index* of the renderer's pipeline state. The marker bundle index is used to select an entry of the renderer's *marker bundle* whenever a bundlable marker attribute's ASF is *Bundled.*

Marker Bundle Index is a member of the Marker attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Marker Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 5 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | marker bundle table index |
| 2 | | unused |

### 6.9.55 Marker Color 5.0

*color*: COLOR_SPECIFIER

This command sets the *marker color* entry of the renderer’s pipeline state. This entry defines the marker color if the pipeline state’s *marker color ASF* is *Individual*.

Output command Marker Color Index can alternately be used to set a marker color of type *Indexed*.

If the color type given is not supported, a color of type *Indexed* with a value of 1 is used.

Marker Color is a member of the Marker attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Marker Color OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 4 | output command number |
| 2 | 2+*sc*/4 | output command length |
| 4+*sc* | COLOR_SPECIFIER | marker color (see COLOR for *sc*) |

### 6.9.56 Marker Color Index

5.0

*index*: TABLE_INDEX

This command sets the *marker color* entry of the renderer's pipeline state. This command is used when the color is of type *Indexed*. The *marker color* entry defines the marker color if the pipeline state's *marker color ASF* is *Individual*.

Output command Marker Color, which allows a color of any type to be specified, can alternately be used.

Marker Color Index is a member of the Marker attribute set.

An *OutputCommand* error is issued if the index is 65535.

**Encoding: Marker Color Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 3 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | indexed marker color |
| 2 | | unused |

### 6.9.57 Marker Scale Factor

5.0

*scale*: FLOAT

This command sets the *marker scale factor* entry of the renderer's pipeline state. This entry defines the marker scale factor if the pipeline state's *marker scale factor ASF* is *Individual*. When a marker primitive is rendered, the nominal marker size is multiplied by the marker scale factor. The result is mapped to the nearest marker size the server can render, so a negative marker scale factor results in the smallest available marker. The *NominalMarkerSize*, *MinMarkerSize*, *MaxMarkerSize,* and *NumSupportedMarkerSizes* are inquired with request **PEXGetImpDepConstants**.

Marker Scale Factor is a member of the Marker attribute set.

**Encoding: Marker Scale Factor OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 2 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | marker scale factor |

### 6.9.58 Marker Type 5.0

*marker_type*: MARKER_TYPE

This command sets the *marker type* entry of the renderer's pipeline state. This entry defines the marker type if the pipeline state's *marker type ASF* is *Individual*. The marker type specifies the shape of a marker primitive. If the renderer's *marker glyph table* entry is Null, *marker_type* selects a member of the *MarkerType* enumeration. The registered values are shown in Table 6-32 . If the value in *marker_type* is not a registered marker type, type *Asterisk* is used. If the renderer's *marker glyph table* entry is non-null, *marker_type* selects an entry of the renderer's marker glyph table.

**Table 6-32 . Enumerated Type *MarkerType***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Dot* | "." which is always displayed as the smallest displayable dot (the *marker scale factor* attribute is ignored) with the dot at the marker position. 5.0 |
| 2 | *Cross* | "+" (cross or plus sign) with intersection at the marker position. 5.0 |
| 3 | *Asterisk* | "*" with intersection at the marker position. 5.0 |
| 4 | *Circle* | "o" with center at marker position. 5.0 |
| 5 | *X* | "x" with intersection at the marker position. 5.0 |

Marker Type is a member of the Marker attribute set.

**Encoding: Marker Type OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 1 | output command number |
| 2 | 2 | output command length |
| 2 | MARKER_TYPE | marker type |
| 2 | | unused |

### 6.9.59 Model Clipping 5.0

*clip_switch*: CARD8

This command sets the *model clip indicator* entry of the renderer's pipeline state. Model clipping involves clipping in model coordinates primitives with the model clipping volume.

Model Clipping is a member of the Global attribute set.

An *OutputCommand* error is reported if the value in *clip_switch* is not *Clip* or *NoClip*.

**Encoding: Model Clipping OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 61 | output command number |
| 2 | 2 | output command length |
| 1 | CARD8 | clip switch |
| | 0 *Clip* 5.0 | |
| | 1 *NoClip* 5.0 | |
| 3 | | unused |

### 6.9.60 Model Clipping Volume

*operator*: MODEL_CLIP_OPERATOR

*halfspaces*: LISTofHALF_SPACE

This command sets the *model clip volume* of the renderer's pipeline state. Primitives outside the model clipping volume are not rendered when the pipeline state's' *model clip indicator* is *Clip.*

The model clip volume is formed with half-spaces. Each half-space is defined by a point in modeling coordinates and a vector defining the normal to the plane which is the boundary of the half-space. The normal points in the direction of the half-space, and the point is considered to be on the boundary plane. If no half-spaces are given, it is assumed the missing half-spaces define a volume which is all of modeling coordinate space.

Each half-space is transformed by the pipeline state's composite modeling transformation and combined with the existing pipeline state's *model clip volume* using the operator specified by *operator.* The registered model clip operators are defined in Table 6-33 . The model clip volume which results is not affected by subsequent changes to the composite modeling transformation.

**Table 6-33 . Enumerated Type *ModelClipOperator***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Replace* | The specified half-spaces are used to create a new composite modeling clipping volume that replaces the current composite modeling clipping volume. 5.0 |
| 2 | *Intersection* | The specified half-spaces are intersected with the existing composite modeling clipping volume to compute a new composite modeling clipping volume. 5.0 |

If the specified operator is not supported by the PEX server, if any half-space is degenerate, or if modifying the pipeline state's *model clip volume* will cause the maximum number of model clipping planes the server can support to be exceeded, the pipeline state's *model clip volume* is not modified. *MaxModelClipPlanes* can be inquired with the **PEXGetImpDepConstants** request.

Model Clipping Volume is a member of the Global attribute set.

**Encoding: Model Clipping Volume OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 62 | output command number |
| 2 | 2+3**fp***n*/2 | output command length |
| 2 | MODEL_CLIP_OPERATOR | model clipping volume operator |
| 2 | CARD16 | number of half-spaces (*n*) |
| *n**6**fp* | LISTofHALF_SPACE | half-spaces |

### 6.9.61 Model Clipping Volume 2D

5.0

*operator*: MODEL_CLIP_OPERATOR

*halfspaces*: LISTofHALF_SPACE_2D

This command sets the *model clip volume* entry of the renderer’s pipeline state. The Z-component of the point and vector pairs defining the modeling coordinates half-spaces are assumed to be 0. Otherwise this output command behaves exactly as the Model Clipping Volume output command.

**Encoding: Model Clipping Volume 2D OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 63 | output command number |
| 2 | 2+*fp***n* | output command length |
| 2 | MODEL_CLIP_OPERATOR | model clipping volume operator |
| 2 | CARD16 | number of half-spaces (*n*) |
| *n**4**fp* | LISTofHALF_SPACE_2D | half-spaces |

### 6.9.62 Parametric Surface Bundle Index

*index*: TABLE_INDEX

This command sets the *parametric surface bundle index* of the renderer's pipeline state. The parametric surface bundle index is used to select an entry of the renderer's *interior bundle* whenever a bundlable parametric surface attribute's ASF is *Bundled* and the renderer's *PHIGS interior bundle mode* is *True.* (If the renderer's *PHIGS interior bundle mode* is *False*, the pipeline state's *interior bundle index* is used instead and the *parametric surface bundle index* is ignored.) The bundlable parametric surface attributes are surface approximation and parametric surface characteristics.

Parametric Surface Bundle Index is a member of the Curved Surface and Parametric Surface attribute sets.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Parametric Surface Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 128 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | interior bundle index |
| 2 | | unused |

### 6.9.63 Parametric Surface Characteristics 5.0

*psc*: PSURF_CHAR

This command sets the *parametric surface characteristics* entry of the renderer's pipeline state. This entry defines the parametric surface characteristics if the pipeline state's *parametric surface characteristics ASF* is *Individual*. The parametric surface characteristics consist of aids in visualizing the shape of a parametric surface. The registered parametric surface characteristics types are defined in Table 6-25 . If the type given in *psc* is not supported by the server, type *None* is used.

**Table 6-34 . Enumerated Type *ParametricSurfaceCharacteristics***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *None* | No special aids in visualizing the shape of a parametric surface are applied.<br>No data is defined for this type. 5.0 |
| 2 | *(implementation-dependent)* | An implementation-dependent aid in visualizing the shade of a parametric surface is applied. This type does not distinguish back-facing portions of the surface. The appearance of the representation is controlled by the appropriate set of primitive attributes for the representation. It is implementation-dependent how the representation interacts with any interior rendering indicated by the interior attributes.<br>No data is defined for this type. 5.0 |
| 3 | *IsoparametricCurves* | Isoparametric curves are drawn on the surface.<br>The data record contains the number of curves to draw in each of the parameter dimensions and their placement. If the placement is *Uniform*, the specified number of curves are evenly spaced between the parameter limits of the surface; curves are also drawn at the parameter limits. If the placement is *NonUniform*, the specified number of curves are evenly spaced between each pair of knots; curves are also drawn at the knots. In both cases only the portions of isoparametric curves are drawn that are within the interior of the surface as defined by any trimming curves.<br>This method does not distinguish back-facing portions of the surface. The tessellation and appearance of the isoparametric curves are controlled by the surface approximation criteria and the polyline attributes, respectively. The isoparametric curves are drawn in addition to any interior rendering indicated by the interior style or back interior style attributes. Isoparametric curves have higher visual priority than the primitive's filled or hollow interiors, but lower priority than the primitive's edges.<br>An *OutputCommand* error is issued if the placement is not *Uniform* or *NonUniform*. 5.0 |

**Table 6-34 . Enumerated Type *ParametricSurfaceCharacteristics* (*Continued*)**

| Value | Mnemonic | Description |
|---|---|---|
| 4 | *MC_LevelCurves* | Level curves are drawn on the surface. The curves correspond to the intersections of the surface and a finite set of planes perpendicular to a modeling coordinate direction vector. The positions of the planes are specified by a sequence of intersection points along an infinite line defined by a modeling coordinate origin point, $P_0$, and a direction vector, $\vec{V}$. <br> $P_i = P_0 - t_i\vec{V}$ **Equation 6.9** <br> The $t_i$ are a sequence of parameters specifying the intersection points. They are in the range: $-\infty < t_i < \infty$. The $P_i$ are the intersection points of the perpendicular planes with the infinite line. $P_0$ is a specified origin point in modeling coordinates, and $\vec{V}$ is the specified direction vector in modeling coordinates. The *i-th* plane is perpendicular to the direction vector, $\vec{V}$, and intersects the infinite line at point $P_i$. <br> This method does not distinguish back-facing portions of the surface. The tessellation and appearance of the level curves are controlled by the surface approximation criteria and the polyline attributes, respectively. The curves are drawn in addition to any interior rendering indicated by the interior style or back interior style attributes. Level curves have higher visual priority than a primitive's filled or hollow interiors, but lower priority than its edges. <br> An *OutputCommand* error is issued if the direction vector has a length of 0. 5.0 |
| 5 | *WC_LevelCurves* | Level curves are drawn on the surface. This type is identical to type *MC_LevelCurves* except the origin point, $P_0$, and the direction vector, $\vec{V}$, are defined in world coordinates instead of modeling coordinates. 5.0 |

Parametric Surface Characteristics is a member of the Parametric Surface attribute set.

An *OutputCommand* error is issued if the data record size doesn't match the parametric surface characteristics type.

**Encoding: Parametric Surface Characteristics OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 72 | output command number |
| 2 | 2+(*n*+*p*)/4 | output command length |
| 4+*n*+*p* | PSURF_CHAR | parametric surface characteristics (see PSURF_CHAR for *n* and *p*) |

### 6.9.64 Pattern Reference Point 5.0

*point*: COORD_2D

This command sets the *pattern reference point, pattern reference vector1,* and *pattern reference vector2* entries of the renderer’s pipeline state. The command is a 2D form of the Pattern Reference Point and Vectors output command (§6.9.65), and differs from it only by containing a 2D value for the pattern reference point defined through *point* and having implicitly defined pattern reference vectors. The Z-coordinate of the reference point is implicitly zero. The first pattern reference vector is implicitly <1,0,0>, and the second pattern reference vector is implicitly <0,1,0>. Once these implicit values are associated with the attribute, it is processed exactly like the Pattern Reference Point and Vectors output command.

**Encoding: Pattern Reference Point OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 47 | output command number |
| 2 | 1+*fp*/2 | output command length |
| 2**fp* | COORD_2D | pattern reference point |

### 6.9.65 Pattern Reference Point And Vectors

5.0

*ref_pt*: COORD_3D

*vector1*: VECTOR_3D

*vector2*: VECTOR_3D

This command sets the *pattern reference point, pattern reference vector1,* and *pattern reference vector2* entries of the renderer's pipeline state. Inquire the *InteriorStyle* enumerated type with request **PEXGetEnumeratedTypeInfo** to determine if interior style *Pattern* is supported by the server. If it is supported, an entry of the renderer's pattern table is selected by the pipeline state's *interior style index* and is applied to fill area primitives of interior style *Pattern*.

The pattern reference point and vectors are defined in modeling coordinates. When applying interior style *Pattern* to a fill area primitive, the reference point and vectors are projected along a normal vector onto the plane of each facet of the primitive. A parallelogram is formed with its origin defined by the projected reference point. One side of the parallelogram is defined by a vector whose direction is parallel to the first projected reference vector and whose length is equal to the absolute value of the X-component of the pattern size. The adjacent side of the parallelogram is defined similarly by the second projected reference vector and the absolute value of the Y-component of the pattern size. The pattern, a list of *m* x *n* equally-sized color cells, is applied across the parallelogram such that the first cell has the parallelogram origin in one corner. The first *m* colors in the pattern correspond to the row of cells nearest to and along the first projected reference vector. The pattern is mapped onto the facet by replicating the pattern parallelogram in directions parallel to the parallelogram's sides until the entire facet is covered.

If at least one of *vector1* or *vector2* is zero length or if the vectors are parallel, this output command is ignored (the pattern entries of the pipeline state are not updated).

Pattern Reference Point and Vectors is a member of the Styled and Shaded Surface attribute set.

**Encoding: Pattern Reference Point and Vectors OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 48 | output command number |
| 2 | 1+9*fp/4 | output command length |
| 3*fp | COORD_3D | pattern reference point |
| 3*fp | VECTOR_3D | pattern reference vector 1 |
| 3*fp | VECTOR_3D | pattern reference vector 2 |

### 6.9.66 Pattern Size

5.0

*size*: VECTOR_2D

This command sets the *pattern size* entry of the renderer's pipeline state. Inquire the *InteriorStyle* enumerated type with request **PEXGetEnumeratedTypeInfo** to determine if interior style *Pattern* is supported by the server. If it is supported, an entry of the renderer's pattern table is selected by the pipeline state's *interior style index* and is applied to fill area primitives of interior style *Pattern*.

The pattern size is defined in modeling coordinates. When applying interior style *Pattern* to a fill area primitive, the reference point and vectors are projected along a normal vector onto the plane of each facet of the primitive. A parallelogram is formed with its origin defined by the projected reference point. One side of the parallelogram is defined by a vector whose direction is parallel to the first projected reference vector and whose length is equal to the absolute value of the X-component of the pattern size. The adjacent side of the parallelogram is defined similarly by the second projected reference vector and the absolute value of the Y-component of the pattern size. The pattern, a list of $m$ x $n$ equally-sized color cells, is applied across the parallelogram such that the first cell has the parallelogram origin in one corner. The first $m$ colors in the pattern correspond to the row of cells nearest to and along the first projected reference vector. The pattern is mapped onto the facet by replicating the pattern parallelogram in directions parallel to the parallelogram's sides until the entire facet is covered.

If the X-component and/or the Y-component of *size* is 0, this output command is ignored (the *pattern size* entry of the pipeline state is not updated).

Pattern Size is a member of the Styled and Shaded Surface attribute set.

**Encoding: Pattern Size OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 46 | output command number |
| 2 | 1+*fp*/2 | output command length |
| 2**fp* | VECTOR_2D | pattern size |

### 6.9.67 Plane Mask

*plane_mask*: BITMASK

This command sets the *plane mask* entry of the renderer's pipeline state to the value given in *plane_mask*. The bits set to 1 in the plane mask determine which planes of the destination image buffer are effected by rendering requests. Each bit of the plane bitmask represents one plane of the image buffer. The least-significant bit (0x00000001) of the plane bitmask corresponds to the plane of a monochrome image buffer; additional planes are represented by the more significant bits. **AllPlanes**, a value defined by X, can be used to refer to all the planes.

The plane mask is intended to function like the core X GC plane mask.

Enabling less than all the planes interacts with alpha blending, primitive antialiasing, transparency, and drawing functions other than *GXcopy* in an implementation-dependent manner.

Plane Mask is a member of the Global attribute set.

**Encoding: Plane Mask OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 129 | output command number |
| 2 | 2 | output command length |
| 4 | BITMASK | plane mask |

### 6.9.68 Polyline Interpolation Method

*polyline_interp*: POLYLINE_INTERP

This command sets the *polyline interpolation method* entry of the renderer's pipeline state. This entry defines the polyline interpolation method if the pipeline state's *polyline interpolation method ASF* is *Individual*. The polyline interpolation method specifies how color values are computed along the segments when rendering polyline primitives. The interpolation method applies to all components of color, including alpha. The registered values for the polyline interpolation method are shown in Table 6-43 . If the value given in *polyline_interp* is not supported by the server, method *None* is used.

**Table 6-35 . Enumerated Type *PolylineInterpMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *None* | No interpolation will be performed between polyline vertices. If color values are supplied that differ for the endpoints of a polyline segment, it is implementation-dependent whether the color of the *i*th vertex will be used to draw the line between the *ith* and *(i+1)th* vertices (if *n* is the number of vertices, the color at the *n*th vertex will be ignored), or whether they will be used to compute an average color which will be applied to the entire segment. 5.0 |
| 2 | *Color* | The polyline's vertex colors (if present) are used. Color values along each polyline segment are then computed by linearly interpolating between the color values at the vertices. 5.0 |

Polyline Interpolation Method is a member of the Shaded Line attribute set.

**Encoding: Polyline Interpolation Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 27 | output command number |
| 2 | 2 | output command length |
| 2 | POLYLINE_INTERP | polyline interpolation method |
| 2 | | unused |

### 6.9.69 Primitive Antialiasing

*prims*: PRIM_AA_BITMASK

*blend_func*: PRIM_AA_BLEND_FUNC

This command sets the *primitive antialiasing* entry of the renderer's pipeline state. Primitive antialiasing specifies modifications made to the image on a type of primitive basis to correct for aliasing effects. Typically the algorithms smooth edges through a blending operation.

The registered primitive types, for the purpose of antialiasing, are listed in Table 6-36 . To determine which primitive types can be antialiased, fetch *PrimAntiAliasSupported* with request **PEXGetImpDepConstants**. The server antialiases any primitives of the types selected by *prims.* It leaves aliased any primitives of the types selected by *prims* which it cannot antialias.

**Table 6-36 . Primitive Types (for Antialiasing) Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0x0001 | *point* | The set of primitives and attributes which combine to form a *point* primitive is implementation-dependent but at a minimum includes marker type *Dot* applied to Marker, Marker 2D, and Marker with Data. |
| 0x0002 | *vector* | The set of primitives and attributes which combine to form a *vector* primitive is implementation-dependent but at a minimum includes Polyline, Polyline 2D, and Polyline Set with Data (Versions 5.0 and 5.2), all of nominal width. |
| 0x0004 | *polygon* | The set of primitives and attributes which combine to form a *polygon* primitive is implementation-dependent but at a minimum includes interior style *Solid* applied to:<br>Arc Area Set with Data<br>Cone Set with Data<br>Fill Area<br>Fill Area 2D<br>Fill Area Set<br>Fill Area Set 2D<br>Fill Area Set with Data (Versions 5.0 and 5.2)<br>Non-uniform B-spline Surface<br>Non-uniform B-spline Surface with Data<br>Sphere Set with Data<br>Quadrilateral Mesh with Data (Versions 5.0 and 5.2)<br>Set of Fill Area Sets with Data (Versions 5.0 and 5.2)<br>Triangle Fan with Data<br>Triangle Set with Data<br>Triangle Strip with Data (Versions 5.0 and 5.2)<br>Triangle Strip Fan Set with Data |
| 0x0008 | *raster_text* | Text, Text 2D, Annotation Text, and Annotation Text 2D with a raster font applied. |
| 0x0010 | *vector_text* | Text, Text 2D, Annotation Text, and Annotation Text 2D with a stroked font applied. |
| 0x0020 | *polygonal_text* | Text, Text 2D, Annotation Text, and Annotation Text 2D with a polygonal font applied. |

If the server is accomplishing primitive antialiasing through a blending function, control over how the blending is accomplished is provided. The defined values for the blending functions are listed in Table 6-37 . If the value given in *blend_func* is not supported by the server, blending function 0 (implementation-dependent) is used.

**Table 6-37 . Enumerated Type *PrimAntiAliasBlendFunction***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *(implementation-dependent)* | The blending function is implementation-dependent. |
| 1 | *SimpleAlpha* | An alpha value is calculated based on the primitive's coverage of the pixel and applied such that: $\text{Color} = \alpha * \text{Color}_{\text{primitive}} + (1 - \alpha) * \text{Color}_{\text{previous}}$ |

Primitive antialiasing interacts with HLHSR, alpha blending, transparency, drawing functions other than *GXcopy*, and plane masking other than all planes enabled in an implementation-dependent manner.

Primitive Antialiasing is a member of the Global attribute set.

An *OutputCommand* error is issued if an undefined bit of the antialiasing primitive bitmask is set.

**Encoding: Primitive Antialiasing OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 130 | output command number |
| 2 | 2 | output command length |
| 2 | PRIM_AA_BITMASK | primitive types to antialias |
| 2 | PRIM_AA_FLEND_FUNC | primitive antialiasing blending function |

### 6.9.70 Reflectance Bundle Index

*index*: TABLE_INDEX

This command sets the *reflectance bundle index* of the renderer's pipeline state. The reflectance bundle index is used to select an entry of the renderer's *interior bundle* whenever a bundlable reflectance attribute's ASF is *Bundled* and the renderer's *PHIGS interior bundle mode* is *True*. (If the renderer's *PHIGS interior bundle mode* is *False*, the pipeline state's *interior bundle index* is used instead and the *reflectance bundle index* is ignored.) The bundlable reflectance attributes are surface reflection model and surface reflection attributes.

Reflectance Bundle Index is a member of the General Surface attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Reflectance Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 131 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | interior bundle index |
| 2 | | unused |

### 6.9.71 Remove Names from Name Set

5.0

*names*: LISTofNAME

This command removes the names in *names* from the *name set* entry of the renderer's pipeline state. Functionalities such as highlighting, invisibility, and picking use filters of included and excluded names to determine whether the functionality should be applied to a primitive based on the pipeline state's *name set*.

Any name in *names* which is above the maximum supported value for a name is ignored. *MaxNamesetNames* can be inquired with request **PEXGetImpDepConstants**.

Remove Names from Name Set is a member of the Global attribute set.

**Encoding: Remove Names From Name Set OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 74 | output command number |
| 2 | 1+*n* | output command length |
| 4**n* | LISTofNAME | list of names (length implicit) |

### 6.9.72 Rendering Color Model 5.0

*model*: RENDERING_COLOR_MODEL

This command sets the *rendering color model* entry of the renderer's pipeline state. The rendering color model defines the color model actually used for the rendering calculations. Color values may be defined in color models other than the rendering color model; they are converted to the rendering color model before being used. The defined values for the rendering color model are shown in Table 6-38 . If the value given in *model* is not supported by the server, rendering color model 0 (implementation-dependent) is used.

**Table 6-38 . Enumerated Type *RenderingColorModel***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *(implementation-dependent)* | An implementation-dependent color space. 5.0 |
| 1 | *RGB* | Red, green, blue color model. 5.0 |
| 2 | *CIE* | CIELUV diagram u,v coordinates plus luminance color model. 5.0 |
| 3 | *HSV* | Hue, saturation, value color model. 5.0 |
| 4 | *HLS* | Hue, lightness, saturation color model. 5.0 |
| 5 | *YIQ* | (NTSC) luminance (Y), inphase (wideband orange-cyan), and quadrature (narrowband magenta-green) color model. 5.2 |

Rendering Color Model is a member of the Global attribute set.

**Encoding: Rendering Color Model OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 71 | output command number |
| 2 | 2 | output command length |
| 2 | RENDERING_COLOR_MODEL | rendering color model |
| 2 | | unused |

### 6.9.73 Restore Model Clipping Volume

5.0

This command sets the *model clip volume* of the renderer’s pipeline state. The *model clip volume* is restored to the value that was inherited by the structure being traversed. If a root structure is being traversed, the *model clip volume* is set to the pipeline state’s default *model clip volume.* Primitives outside the model clipping volume are not rendered when the pipeline state’s’ *model clip indicator* is *Clip.*

Restore Model Clipping Volume is a member of the Global attribute set.

**Encoding: Restore Model Clipping Volume OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 64 | output command number |
| 2 | 1 | output command length |

### 6.9.74 Surface Active Textures

5.2

*active_textures*: LISTofTABLE_INDEX

This command replaces the values in the *active textures* entry of the renderer's pipeline state. The ordered list of indices into the renderer's texture mapping lookup table selects the textures that are sequentially applied to surfaces whose interior style is *Texture*.

If the number of indices given exceeds the maximum number of simultaneously supported texture maps, those textures beyond the limit will be ignored. Inquire *MaxTextureMaps* with request **PEXGetImpDepConstants** to determine this limit.

Surface Active Textures is a member of the Styled and Shaded Surface attribute set.

An *OutputCommand* error is issued if a texture mapping lookup table index of 0 is in the list.

**Encoding: Surface Active Textures OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 142 | output command number |
| 2 | 2+(2*$n$+$p$)/4 | output command length |
| 2 | CARD16 | number of active textures ($n$) |
| 2 | | unused |
| 2*$n$ | LISTofTABLE_INDEX | prioritized active textures |
| $p$ | | $p$=pad(2*$n$) |

### 6.9.75 Surface Approximation Method 5.0

*approx*: SURFACE_APPROX

This command sets the *surface approx method* entry of the renderer's pipeline state. This entry defines the surface approximation if the pipeline state's *surface approx method ASF* is *Individual*. The surface approximation specifies the smoothing method to be used when rendering curved bivariate primitives. The defined surface approximation methods and their usage of the *u_tolerance* and *v_tolerance* parameters are defined in Table 6-24 .

**Table 6-39 . Enumerated Type *SurfaceApproxMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | (implementation-dependent) | The surface is smoothed in an implementation-dependent manner. *u_tolerance* and *v_tolerance* are ignored. 5.0 |
| 2 | *ConstantBetweenKnots* | The surface is tessellated with equal parametric increments between successive pairs of knots.<br>The *u_tolerance* and *v_tolerance* fields control the tessellation in each of the two parameter dimensions. If a tolerance does not define an integer value, it is truncated and only the integer portion is used. If *a tolerance* is less than or equal to zero, the surface will be evaluated only at the parameter limits and at the knots that are within the specified parameter range in that tolerance's direction. If *a tolerance* is greater than zero, in the tolerance's direction, the surface will be evaluated at the parameter limits, at the knots that are within the specified parameter range, and at the number of positions specified by the *tolerance* between each pair of knots. 5.0 |
| 3 | *WCS_ChordalSize* | The surface is tessellated such that the WC length of each segment (chord) in the *u* direction and each chord in the *v* direction is less than the *u_tolerance* and *v_tolerance* (specified in WCs), respectively. 5.0 |
| 4 | *NPC_ChordalSize* | The surface is tessellated such that the NPC length of each segment (chord) in the *u* direction and each chord in the *v* direction is less than the *u_tolerance* and *v_tolerance* (specified in NPCs), respectively. 5.0 |
| 5 | *DC_ChordalSize* | The surface is tessellated such that the DC length of each segment (chord) in the *u* direction and each chord in the *v* direction is less than the *u_tolerance* and *v_tolerance* (specified in DCs), respectively. 5.0 |
| 6 | *WCS_PlanarDev* | The surface is tessellated such that the deviation in WCs between the surface and its tessellated representation in the *u* direction and the surface and its tessellated representation in the *v* direction does not exceed the *u_tolerance* and *v_tolerance* (specified in WCs), respectively. 5.0 |
| 7 | *NPC_PlanarDev* | The surface is tessellated such that the deviation in NPCs between the surface and its tessellated representation in the *u* direction and the surface and its tessellated representation in the *v* direction does not exceed the *u_tolerance* and *v_tolerance* (specified in NPCs), respectively. 5.0 |
| 8 | *DC_PlanarDev* | The surface is tessellated such that the deviation in DCs between the surface and its tessellated representation in the *u* direction and the surface and its tessellated representation in the *v* direction does not exceed the *u_tolerance* and *v_tolerance* (specified in DCs), respectively. 5.0 |

**Table 6-39 . Enumerated Type *SurfaceApproxMethod* (Continued)**

| Value | Mnemonic | Description |
|---|---|---|
| 9 | *WCS_Relative* | *u_tolerance* is a value in the range [0.0, 1.0] which indicates a relative quality of rendering to be maintained independent of scaling in WCs. A value of 1.0 indicates the highest possible quality. *v_tolerance* is ignored. 5.0 |
| 10 | *NPC_Relative* | *tolerance* is a value in the range [0.0, 1.0] which indicates a relative quality of rendering to be maintained independent of scaling in NPCs. A value of 1.0 indicates the highest possible quality. *v_tolerance* is ignored. 5.0 |
| 11 | *DC_Relative* | *tolerance* is a value in the range [0.0, 1.0] which indicates a relative quality of rendering to be maintained independent of scaling in DCs. A value of 1.0 indicates the highest possible quality. *v_tolerance* is ignored. 5.0 |

Specification of lower relative approximation quality is intended to result in smoothing at relatively higher speeds. Likewise specification of relatively higher quality is intended to result in smoothing at possibly lower speeds. Implementations may smooth with a higher quality than requested.

If the surface approximation method is not one of the registered values, method 1 (implementation-dependent) is used. If a specified tolerance cannot be achieved, the most accurate value available for the method is used.

Surface Approximation Method is a member of the Curved Surface attribute set.

**Encoding: Surface Approximation Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 43 | output command number |
| 2 | 2+*fp*/2 | output command length |
| 4+2**fp* | SURFACE_APPROX | surface approximation method and tolerances |

### 6.9.76 Surface Color

5.0

*color*: COLOR_SPECIFIER

This command sets the *surface color* entry of the renderer’s pipeline state. This entry defines the surface color if the pipeline state’s *surface color ASF* is *Individual*.

Output command Surface Color Index can alternately be used to set a surface color of type *Indexed*.

If the color type given is not supported, a color of type *Indexed* with a value of 1 is used.

Surface Color is a member of the Styled and Shaded Surface attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Surface Color OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 32 | output command number |
| 2 | 2+*sc*/4 | output command length |
| 4+*sc* | COLOR_SPECIFIER | surface color (see COLOR for *sc*) |

### 6.9.77 Surface Color Index

5.0

*index*: TABLE_INDEX

This command sets the *surface color* entry of the renderer's pipeline state. This command is used when the color is of type *Indexed*. The *surface color* entry defines the surface color if the pipeline state's *surface color ASF* is *Individual*.

Output command Surface Color, which allows a color of any type to be specified, can alternately be used.

Surface Color Index is a member of the Styled and Shaded Surface attribute set.

An *OutputCommand* error is issued if the index is 65535.

**Encoding: Surface Color Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 31 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | indexed surface color |
| 2 | | unused |

### 6.9.78 Surface Edge Bundle Index

5.0

*index*: TABLE_INDEX

This command sets the *edge bundle index* of the renderer's pipeline state. The edge bundle index is used to select an entry of the renderer's *edge bundle* whenever a bundlable surface edge attribute's ASF is *Bundled.*

Surface Edge Bundle Index is a member of the Edge attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Surface Edge Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 55 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | surface edge bundle table index |
| 2 | | unused |

### 6.9.79 Surface Edge Color

5.0

*color*: COLOR_SPECIFIER

This command sets the *surface edge color* entry of the renderer's pipeline state. This entry defines the surface edge color if the pipeline state's *surface edge color ASF* is *Individual*.

Output command Surface Edge Color Index can alternately be used to set a surface edge color of type *Indexed*.

If the color type given is not supported, a color of type *Indexed* with a value of 1 is used.

Surface Edge Color is a member of the Edge attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Surface Edge Color OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 54 | output command number |
| 2 | 2+*sc*/4 | output command length |
| 4+*sc* | COLOR_SPECIFIER | surface edge color (see COLOR for *sc*) |

### 6.9.80 Surface Edge Color Index

5.0

*index*: TABLE_INDEX

This command sets the *surface edge color* entry of the renderer's pipeline state. This command is used when the color is of type *Indexed*. The *surface edge color* entry defines the surface edge color if the pipeline state's *surface edge color ASF* is *Individual*.

Output command Surface Edge Color, which allows a color of any type to be specified, can alternately be used.

Surface Edge Color Index is a member of the Edge attribute set.

An *OutputCommand* error is issued if the index is 65535.

**Encoding: Surface Edge Color Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 53 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | indexed surface edge color |
| 2 | | unused |

### 6.9.81 Surface Edge Flag

5.0

*onoff*: SWITCH

This command sets the *surface edges* entry of the renderer's pipeline state. The *surface edges* entry defines surface edging if the pipeline state's *surface edges ASF* is *Individual*. When surface edging is enabled, edges are rendered on the surface primitives with the surface edge attributes. Otherwise the surface edges are not rendered with surface edge attributes.

Surface Edge Flag is a member of the Edge attribute set.

An *OutputCommand* error is issued if the value in *onoff* is not *On* or *Off*.

**Encoding: Surface Edge Flag OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 50 | output command number |
| 2 | 2 | output command length |
| 1 | SWITCH | surface edges enabled / disabled |
| 3 | | unused |

### 6.9.82 Surface Edge Type

5.0

*edge_type*: SURFACE_EDGE_TYPE

This command sets the *surface edge type* entry of the renderer's pipeline state. This entry defines the surface edge type if the pipeline state's *surface edge type ASF* is *Individual*. The surface edge type describes the style used when rendering edges on surface primitives. The registered values for the surface edge type are defined in Table 6-40 . If the value given for the surface edge type is not supported by the server, type *Solid* is used.

**Table 6-40 . Enumerated Type *SurfaceEdgeType***

| Value | Mnemonic | Description†1 |
|---|---|---|
| 1 | *Solid* | Draw the surface edge with a solid, unbroken line. |
| 2 | *Dashed* | Draw the surface edge with a line that is dashed. |
| 3 | *Dotted* | Draw the surface edge with a line that is dotted. |
| 4 | *DashDot* | Draw the surface edge with a line that contains alternating dots and dashes. |

†1) It is implementation-dependent whether the sequence for the *Dashed, Dotted,* and *DashDot* surface edge types is restarted or continued at the start of the edge, at the start of a clipped segment of an edge, and at each vertex.

Surface Edge Type is a member of the Edge attribute set.

**Encoding: Surface Edge Type OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 51 | output command number |
| 2 | 2 | output command length |
| 2 | SURFACE_EDGE_TYPE | surface edge type |
| 2 | | unused |

### 6.9.83 Surface Edge Width

*width*: FLOAT

This command sets the *surface edge width* entry of the renderer's pipeline state. This entry defines the surface edge width factor if the pipeline state's *surface edge width ASF* is *Individual*. When a surface primitive edge is rendered, the nominal edge width is multiplied by the surface edge width factor. The result is mapped to the nearest edge width the server can render, so a negative surface edge width factor results in the thinnest available edge. The *NominalEdgeWidth*, *MinEdgeWidth*, *MaxEdgeWidth,* and *NumSupportedEdgeWidths* can be inquired with request **PEXGetImpDepConstants**.

Surface Edge Width is a member of the Edge attribute set.

**Encoding: Surface Edge Width OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 52 | output command number |
| 2 | 1+*fp*/4 | output command length |
| *fp* | FLOAT | surface edge width factor |

### 6.9.84 Surface Interior Style 5.0

*interior_style*: INTERIOR_STYLE

This command sets the *interior style* entry of the renderer’s pipeline state. This entry defines the interior style if the pipeline state’s *interior style ASF* is *Individual*. The interior style describes how the interior of surfaces is “filled”. The registered values for the interior style are defined in Table 6-41 .

**Table 6-41 . Enumerated Type *InteriorStyle***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Hollow* | The interiors of surface primitives are not filled, but the boundary is drawn using the surface color. If the surface primitive is clipped as a result of modeling, view, or workstation clipping, the boundary must be drawn along the clipped boundary as well. 5.0 |
| 2 | *Solid* | The interiors of surface primitives are filled using the surface color. 5.0 |
| 3 | *Pattern* | The interiors of surface primitives are filled with a pattern defined by the entry of the renderer’s *pattern table* specified by the pipeline state’s *interior style index* or *bf interior style index*. 5.0 |
| 4 | *Hatch* | The interiors of surface primitives are filled using the surface color and the hatch style whose index is specified by the pipeline state’s *interior style index* or *bf interior style index*. 5.0 |
| 5 | *Empty* | The interiors of the surface primitives are not drawn at all. 5.0 |
| 6 | *Texture* | The interiors of surface primitives are textured using the texture maps specified by the pipeline state’s *active textures*. If the pipeline state’s *facet distinguish flag* is *True*, the pipeline state’s *bf active textures* is used to select the texture maps for back-facing surfaces. 5.2 |

Surface Interior Style is a member of the Styled and Shaded Surface attribute set.

If the value given for the interior style is not one of the registered interior styles, an *OutputCommand* error is issued. If the value given for the interior style is not supported by the server, style *Hollow* is used.

**Encoding: Surface Interior Style OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 29 | output command number |
| 2 | 2 | output command length |
| 2 | INTERIOR_STYLE | surface interior style |
| 2 | | unused |

### 6.9.85 Surface Interior Style Index

*index*: TYPE_OR_TABLE_INDEX

This command sets the *interior style index* entry of the renderer's pipeline state. This entry defines the interior style index if the pipeline state's *interior style index ASF* is *Individual*. If the pipeline state's *interior style* is *Pattern* or *Hatch*, the interior style index is used to further define how the interior of surfaces is "filled".

For interior style *Hatch*, the interior style index determines the hatch style. The registered values for hatch styles are defined in Table 6-42 . If the value given in *index* is not supported by the server, style *45Degrees* is used.

**Table 6-42 . Enumerated Type *HatchStyle***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *45Degrees* | Draw the surface interior with single pixel width lines at about 45 degrees in device coordinates. |
| 2 | *135Degrees* | Draw the surface interior with single pixel width lines at about 135 degrees in device coordinates. |

For interior style *Pattern*, the index selects an entry of the renderer's *pattern table*. If the index given in *index* is not supported by the server, index one is used.

Surface Interior Style Index is a member of the Styled and Shaded Surface attribute set.

**Encoding: Surface Interior Style Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 30 | output command number |
| 2 | 2 | output command length |
| 2 | TYPE_OR_TABLE_INDEX | surface interior style index |
| 2 | | unused |

### 6.9.86 Surface Interpolation Method

*surface_interp*: SURFACE_INTERP

This command sets the *surface interpolation method* entry of the renderer’s pipeline state. This entry defines the surface interpolation method if the pipeline state’s *surface interpolation method ASF* is *Individual*. The surface interpolation method specifies how color values are computed in the interior of surface primitives. The interpolation method applies to all components of color, including alpha. The registered values for the surface interpolation method are shown in Table 6-43 . If the value given in *surface_interp* is not supported by the server, method *None* is used.

**Table 6-43 . Enumerated Type *SurfaceInterpMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *None* | The color resulting from a single evaluation of the reflectance calculation is used for the entire surface. No interpolation will be performed across surface interiors and edges. |
| 2 | *Color* | The reflection calculation is performed at the vertices of the surface. The resulting color values are then linearly interpolated across the interior of the surface and the edges. |
| 3 | *DotProduct* | Any dot products needed by the reflectance calculation are computed at each vertex. These dot products are linearly interpolated and the reflectance calculation is performed using these values to compute the color value at each pixel in the interior of a surface or along a surface edge. |
| 4 | *Normal* | The vertex reflectance normals are interpolated and the reflectance calculation is performed at each pixel in the interior of a surface or along a surface edge. |

Surface Interpolation Method is a member of the Styled and Shaded Surface attribute set.

**Encoding: Surface Interpolation Method OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 35 | output command number |
| 2 | 2 | output command length |
| 2 | SURFACE_INTERP | surface interpolation method |
| 2 | | unused |

### 6.9.87 Surface Reflection Attributes

5.0

*attr*: REFLECTION_ATTR

This command sets the *reflection attributes* entry of the renderer's pipeline state. This entry defines the reflection attributes if the pipeline state's *reflection attributes ASF* is *Individual*. The reflection attributes are used in the reflection calculation (§2.2.2.2) used to light surface primitives.

The *ambient coef, diffuse coef,* and *specular coef* entries of the reflection attributes define the fraction of ambient, diffuse, and specular light respectively reflected by a surface. A value of 0.0 means no light is reflected; 1.0 means 100% of the light is reflected.

The larger the value of the *specular conc* field of the reflection attributes, the shinier the surface.

Inquire *TransparencySupported* with request **PEXGetImpDepConstants** to determine whether the transmission coefficient is utilized in the reflection calculation. The *transmission coef* entry of the reflection attributes is used to define the default alpha component of surface primitives without an explicit alpha-component. This alpha value is determined by subtracting the transmission coefficient from 1, since a transmission coefficient of 0.0 means the surface is opaque, while a coefficient of 1.0 means the surface is transparent.

If the color type given is not supported, a specular color of type *Indexed* with a value of 1 is used.

Surface Reflection Attributes is a member of the General Surface attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Surface Reflection Attributes OC**

| **# of Bytes** | **Data Type or Value** | **Description** |
|---|---|---|
| 2 | 33 | output command number |
| 2 | 2+(*sc*+5**fp*)/4 | output command length |
| 4+*sc*+5**fp* | REFLECTION_ATTR | reflection attributes (see COLOR for *sc*) |

### 6.9.88 Surface Reflection Model 5.0

*reflection_model*: REFLECTION_MODEL

This command sets the *reflection model* entry of the renderer’s pipeline state. This entry defines the reflection model if the pipeline state’s *reflection model ASF* is *Individual*. The reflection model specifies which reflectance components are considered when the reflectance calculations are conducted. The registered reflection model values are defined in Table 6-44 . If the value in *reflection _model* is not supported by the server, *None* is used.

**Table 6-44 . Enumerated Type *ReflectionModel***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *None* | No reflectance calculation is performed. The surface color is not affected by light sources. 5.0 |
| 2 | *Ambient* | Only the ambient terms of the reflectance calculation are used. 5.0 |
| 3 | *Diffuse* | Only the ambient and diffuse terms of the reflectance calculated are used. 5.0 |
| 4 | *Specular* | The ambient, diffuse, and specular terms of the reflectance calculation are all used. 5.0 |

Surface Reflection Model is a member of the General Surface attribute set.

**Encoding: Surface Reflection Model OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 34 | output command number |
| 2 | 2 | output command length |
| 2 | REFLECTION_MODEL | surface reflection model |
| 2 | | unused |

### 6.9.89 Text Alignment

*alignment*: TEXT_ALIGNMENT

This command sets the *text alignment* entry of the renderer's pipeline state. The text alignment is used to position the text primitive's extent relative to the text position. The extent is an imaginary parallelogram whose sides are parallel to the X and Y axes of the text's coordinate system. The sides are bounded by the font-specific character bodies of the string's characters. The registered vertical text alignment values are listed in Table 6-45 and the registered horizontal text alignment values are listed in Table 6-46 .

**Table 6-45 . Vertical Text Alignment Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0 | *Normal* | The vertical text alignment is based on the text path as follows:<br>Text Path — Vertical Text Alignment<br>*Right* — *Base*<br>*Left* — *Base*<br>*Up* — *Base*<br>*Down* — *Top* |
| 1 | *Top* | The top of the extent passes through the text position. |
| 2 | *Cap* | If the text path is *Right* or *Left*, the capline of the string passes through the text position. Otherwise the capline of the topmost character in the string passes through the text position.<br>The font-specific capline is the horizontal line which bounds the top of the rendered characters. Typically there is some space between the capline and topline where rendering above the character, such as an accent, would be placed. |
| 3 | *Half* | If the text path is *Right* or *Left*, the halfline of the string passes through the text position. Otherwise the text position lies halfway between the halflines of the top and bottom characters.<br>The font-specific halfline is the horizontal line which lies halfway between the capline and baseline. |
| 4 | *Base* | If the text path is *Right* or *Left*, the baseline of the string passes through the text position. Otherwise the baseline of the bottommost character in the string passes through the text position.<br>The font-specific baseline is the horizontal line upon which the rendered characters sit. Typically there is some space between the baseline and bottomline where for rendering the lower portions of lowercase letters such as "j" and "q". |
| 5 | *Bottom* | The bottom of the extent passes through the text position. |

**Table 6-46 . Horizontal Text Alignment Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0 | *Normal* | The horizontal text alignment is based on the text path as follows:<br>Text Path — Horizontal Text Alignment<br>*Right* — *Left*<br>*Left* — *Right*<br>*Up* — *Center*<br>*Down* — *Center* 5.0 |
| 1 | *Left* | The left side of the extent passes through the text position. 5.0 |
| 2 | *Center* | The text position lies midway between the left and right sides of the extent. 5.0 |
| 3 | *Right* | The right side of the extent passes through the text position. 5.0 |

Text Alignment is a member of the Text attribute set.

An *OutputCommand* error is issued if the horizontal or vertical alignment is not one of the registered values.

**Encoding: Text Alignment OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 15 | output command number |
| 2 | 2 | output command length |
| 4 | TEXT_ALIGNMENT | horizontal and vertical text alignment |

### 6.9.90 Text Bundle Index

5.0

*index*: TABLE_INDEX

This command sets the *text bundle index* of the renderer's pipeline state. The text bundle index is used to select an entry of the renderer's *text bundle* whenever a bundlable text attribute's ASF is *Bundled.*

Text Bundle Index is a member of the Annotation and Text attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Text Bundle Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 21 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | text bundle table index |
| 2 | | unused |

### 6.9.91 Text Color

5.0

*color*: COLOR_SPECIFIER

This command sets the *text color* entry of the renderer’s pipeline state. This entry defines the text color if the pipeline state’s *text color ASF* is *Individual*.

Output command Text Color Index can alternately be used to set a text color of type *Indexed*.

If the color type given is not supported, a color of type *Indexed* with a value of 1 is used.

Text Color is a member of the Annotation and Text attribute set.

Out of range direct color components are treated in an implementation-dependent way. An *OutputCommand* error is issued if the color type is *Indexed* and the color index is 65535.

**Encoding: Text Color OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 11 | output command number |
| 2 | 2+*sc*/4 | output command length |
| 4+*sc* | COLOR_SPECIFIER | text color (see COLOR for *sc*) |

### 6.9.92 Text Color Index 5.0

*index*: TABLE_INDEX

This command sets the *text color* entry of the renderer's pipeline state. This command is used when the color is of type *Indexed*. The *text color* entry defines the text color if the pipeline state's *text color ASF* is *Individual*.

Output command Text Color, which allows a color of any type to be specified, can alternately be used.

Text Color Index is a member of the Annotation and Text attribute set.

An *OutputCommand* error is issued if the index is 65535.

**Encoding: Text Color Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 10 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | indexed text color |
| 2 | | unused |

### 6.9.93 Text Font Index

5.0

*index*: TABLE_INDEX

This command sets the *text font index* of the renderer’s pipeline state. The *text font index* entry defines the text font index if the pipeline state’s *text font index ASF* is *Individual*. The text font index is used to select an entry of the renderer’s *text font table*. Each entry of the text font table defines a font group.

Text Font Index is a member of the Annotation and Text attribute set.

An *OutputCommand* error is issued if the value in *index* is zero.

**Encoding: Text Font Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 6 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | text font table index |
| 2 | | unused |

### 6.9.94 Text Path

*path*: TEXT_PATH

This command sets the *text path* entry of the renderer's pipeline state. The text path defines the direction along which text characters are placed. The registered text paths are listed in Table 6-47 .

**Table 6-47 . Text Path Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0 | *Right* | Each character is placed after the previous character along a baseline in the direction of the character base vector. |
| 1 | *Left* | Each character is placed after the previous character along a baseline in the opposite direction of the character base vector. |
| 2 | *Up* | Each character is placed after the previous character along the direction of the character up vector. |
| 3 | *Down* | Each character is placed after the previous character along the opposite direction of the character up vector. |

Text Path is a member of the Text attribute set.

An *OutputCommand* error is issued if the value in *path* is not one of the registered text path values.

**Encoding: Text Path OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 14 | output command number |
| 2 | 2 | output command length |
| 2 | TEXT_PATH | text path |
| 2 | | unused |

### 6.9.95 Text Precision 5.0

*precision*: TEXT_PRECISION

This command sets the *text precision* entry of the renderer’s pipeline state. The *text precision* entry defines the text precision if the pipeline state’s *text precision ASF* is *Individual*. The text precision defines how precisely the server should apply attributes to and clip text. The registered text precision values are defined in Table 6-48 .

**Table 6-48 . Text Precision Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0 | *String* | The Text Font Index, Text Color, Character Height, and Character Expansion attributes are applied to text primitives.<br>The Character Up Vector, Text Path, Text Alignment, and Character Spacing attributes are not necessarily applied to text primitives.<br>Clipping is implementation-dependent. 5.0 |
| 1 | *Char* | All text attributes are applied to text primitives.<br>Clipping is performed at least on a character by character basis. 5.0 |
| 2 | *Stroke* | All text attributes are applied to text primitives.<br>Clipping is performed exactly at the clipping boundaries. 5.0 |

A server may implement any text precision as a better text precision, where *String* is the worst precision and *Stroke* the best precision.

Some fonts can only be rendered with lesser precisions. All the mono-encodings of a text primitive will be rendered with the same (least common denominator) precision.

Text Precision is a member of the Annotation and Text attribute set.

An *OutputCommand* error is reported if *precision* is not one of the registered text precision values.

**Encoding: Text Precision OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 7 | output command number |
| 2 | 2 | output command length |
| 2 | TEXT_PRECISION | precision |
| 2 | | unused |

### 6.9.96 Texture Mapping Perspective Correction

*tm_perspective_correction*: TM_PERSPECTIVE_CORRECTION

This command sets the *tm perspective correction* entry of the renderer's pipeline state. The texture mapping perspective correction specifies if, and how, texture coordinates are adjusted for a perspective view mapping projection when rendering textured surface primitives. The defined values for texture mapping perspective correction are shown in Table 6-49 . If the value given in *tm_perspective_correction* is not supported by the server, the perspective correction value used is implementation-dependent.

**Table 6-49 . Enumerated Type *TMPerspectiveCorrection***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *None* | Texture coordinates are linearly interpolated without any effort to apply correction or perspective view mapping. |
| 1 | *Vertex* | The texture coordinates associated with each vertex are manipulated to account for perspective view mapping. The corrected values are then linearly interpolated between vertices. |
| 2 | *Pixel* | As texture coordinates are interpolated, their values are manipulated at each step to account for perspective view mapping. |

Texture Mapping Perspective Correction is a member of the Styled and Shaded Surface attribute set.

**Encoding: Texture Mapping Perspective Correction OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 144 | output command number |
| 2 | 2 | output command length |
| 2 | TM_PERSPECTIVE_CORRECTION | perspective correction |
| 2 | | unused |

### 6.9.97 Texture Mapping Resource Hints

*optimize*: TM_RESOURCE_HINTS

*prioritized_textures*: LISTofTABLE_INDEX

This command sets the *tm resource hints* entry of the renderer’s pipeline state. The texture mapping resource hints specify how the server should consider optimizing its resources when subsequent entries of the texture mapping lookup table are activated. The defined values for the texture mapping resource optimization hints are listed in Table 6-50 . If the value given in *optimize* is not supported by the server, resource hint *None* is used.

**Table 6-50 . Enumerated Type *TMResourceHints***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *None* | No optimization hint is provided. |
| 1 | *Speed* | Speed is the most important consideration. |
| 2 | *Space* | Space is the most important consideration. |

*The pipeline state’s tm resource hints* also includes a prioritized list of indices into the renderer’s texture mapping lookup table. This list indicates the expected usage of the indicated textures. If the server uses the list, it is only applied to subsequently activated textures, both when they are activated and when they are deactivated. Textures already activated and subsequently activated textures that are not in the list are assumed to be at some implementation-dependent priority lower than the textures in the list. The first texture in the list is expected to be activated most often, and the last texture the least, of the given textures. If the same index appears multiple times in the list, its highest priority is used. If an index is not defined, it is ignored. The *prioritized_textures* replace the list in the pipeline state’s *tm resource hints*.

Texture Mapping Resource Hints is a member of the Styled and Shaded Surface attribute set.

An *OutputCommand* error is issued if an index in *prioritized_textures* is 0.

**Encoding: Texture Mapping Resource Hints OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 145 | output command number |
| 2 | 2+(2**n*+*p*)/4 | output command length |
| 2 | TM_RESOURCE_HINTS | texture mapping resource hint |
| 2 | CARD16 | number of prioritized textures (*n*) |
| 2**n* | LISTofTABLE_INDEX | prioritized textures |
| *p* | | *p*=pad(2**n*) |

### 6.9.98 Texture Mapping Sample Frequency

*tm_sample_frequency*: TM_SAMPLE_FREQUENCY

This command sets the *tm sample frequency* entry of the renderer's pipeline state. *The texture mapping sample frequency* specifies the frequency to sample texels in a texture map when rendering textured surface primitives. The defined values for texture mapping sample frequency are shown in Table 6-51 . If the value given in *tm_sample_frequency* is not supported by the server, frequency *Pixel* is used.

**Table 6-51 . Enumerated Type *TMSampleFrequency***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *Pixel* | Texture map texels are sampled once for each pixel. |
| 1 | *InterpDep* | The texture map sample frequency is dependent on the pipeline state's *surface interpolation method* as follows:<br>Interpolation Method — *InterpDep* Sample Frequency<br>*SurfaceInterpNone* — per facet<br>*SurfaceInterpColor* — per vertex<br>*SurfaceInterpDot* — per pixel<br>*SurfaceInterpNormal* — per pixel<br>If the pipeline state's *facet distinguish flag* is *True*, the pipeline state's *bf surface interpolation method* is applied for back-facing facets. |

Texture Mapping Sample Frequency is a member of the Styled and Shaded Surface attribute set.

**Encoding: Texture Mapping Sample Frequency OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 141 | output command number |
| 2 | 2 | output command length |
| 2 | TM_SAMPLE_FREQUENCY | texture mapping sampling frequency |
| 2 | | unused |

### 6.9.99 Vertex Order

*vertex order*: VERTEX_ORDER

This command sets the *vertex order* entry of the renderer’s pipeline state. The vertex order is used in computing the facet normals of surfaces. For “with data” area primitives which do not have facet normals supplied and the other area primitives, the normal for each facet is calculated assuming the vertex order given through *vertex order*. (See §2.1.1.1.1 and §2.2.2.2 for information on how surface geometric and reflectance normals are calculated.)

Vertex Order is a member of the General Surface attribute set.

An *OutputCommand* error is reported if *vertex order* is not set to *Righthand* or *Lefthand*.

**Encoding: Vertex Order OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 132 | output command number |
| 2 | 2 | output command length |
| 2 | VERTEX_ORDER | vertex order |
| 2 | | unused |

### 6.9.100 View Index

5.0

*index*: TABLE_INDEX

This command sets the *view index* of the renderer's pipeline state. The view index is used to select an entry of the renderer's *view table*. The view defines a volume in world coordinates being viewed, from where it is viewed, and how it is placed in NPC space.

View Index is a member of the Global attribute set.

An *OutputCommand* error is issued if the value in *index* is 65535.

**Encoding: View Index OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 65 | output command number |
| 2 | 2 | output command length |
| 2 | TABLE_INDEX | view table index |
| 2 | | unused |

### 6.9.101 Z-buffer Test

*z_test*: Z_TEST

This command sets the *Zbuffer test* entry of the renderer’s pipeline state. The Z-buffer test specifies the criterion used to determine whether a pixel is visible or hidden when HLHSR is enabled and accomplished via Z-buffering. *z_test* defines how the pixel’s Z-component is compared against the corresponding Z-buffer value. If the comparison is True, the pixel is visible. Note the Z-buffer test is applied in a right-handed coordinate system where a larger Z-value is closer to the eyepoint. The registered values for the Z-buffer test are defined in §Table 6-43 . If the value given in *z_test* is not supported by the server, the value used is implementation-dependent.

**Table 6-52 . Enumerated Type *ZTest***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *LessThan* | if pixel z value < z-buffer value, pixel is visible |
| 1 | *LessThanOrEqual* | if pixel z value <= z-buffer value, pixel is visible |
| 2 | *GreaterThan* | if pixel z value > z-buffer value, pixel is visible |
| 3 | *GreaterThanOrEqual* | if pixel z value >= z-buffer value, pixel is visible |
| 4 | *Equal* | if pixel z value == z-buffer value, pixel is visible |
| 5 | *NotEqual* | if pixel z value != z-buffer value, pixel is visible |
| 6 | *Pass* | pixel is always visible |

Z-Buffer Test is a member of the Global attribute set.

**Encoding: Z-Buffer Test OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 133 | output command number |
| 2 | 2 | output command length |
| 2 | Z_TEST | Z-buffer test |
| 2 | | unused |

## 6.10 Other Output Commands

This section gives the definitions of output commands which are independent of primitives and their attributes. The majority of these output commands are for traversal flow-of-control or are convenience functions to make structure manipulation easier for the client-application.

### 6.10.1 Application Data

5.0

*data*: data

When included in a **PEXStoreElements** request, this command allows client application data to be stored in a structure. This output command has no affect during rendering.

Byte-swapping and floating-point conversion are not done on this type of output command.

**Encoding: Application Data OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 77 | output command number |
| 2 | 2+(*n*+*p*)/4 | output command length |
| 2 | CARD16 | length of data (*n*) |
| 2 | | unused |
| *n* | | data |
| *p* | | *p*=pad(*n*) |

### 6.10.2 Condition Flags

*enable*: BITMASK

*disable*: BITMASK

This command modifies the value of selected flags bits of the *condition flags* entry of the renderer's pipeline state. The bits set in the *enable* list will force the corresponding bits in *condition flags* to be set to one. The bits set in the *disable* list will force the corresponding bits in *condition flags* to be set to zero.

The pipeline state's *condition flags* are also modified with the Test Extent (§6.10.12) and Test Extend 2D (§6.10.13) OCs. The *condition flags* are used when processing the Conditional Execute Structure, Conditional Include Structure, and Conditional Return OCs.

An *OutputCommand* error is issued if the same condition flag is specified in both *enable* and *disable*.

**Encoding: Condition Flags OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 134 | output command number |
| 2 | 3 | output command length |
| 4 | BITMASK | condition flags to be enabled |
| 4 | BITMASK | condition flags to be disabled |

### 6.10.3 Conditional Execute Structure

| | |
|---|---|
| *mask*: | BITMASK |
| *test*: | CONDITION_TEST |
| *s_id*: | STRUCTURE_ID |

This command transfers flow-of-control to the specified structure if the condition is satisfied. The condition is satisfied if the bits of the pipeline state's *condition flags* corresponding to the bits set in *mask* meet the condition described by *test*. The registered values for *test* are shown in Table 6-53 .

**Table 6-53 . Condition Test Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 0 | *AllEnabled* | All selected bits are set to one. |
| 1 | *AllDisabled* | All selected bits are set to zero. |
| 2 | *NotAllDisabled* | At least one selected bit is set to one. |
| 3 | *NotAllEnabled* | At least one selected bit is set to zero. |
| 4 | *AlwaysPass* | Always pass. |
| 5 | *AlwaysFail* | Always fail. |

If the condition is satisfied, processing of output commands will commence with the first structure element in the structure specified by *s_id*. All the pipeline state attribute values of the renderer are saved. The composite modeling transform is copied into the pipeline state's *global transform* and the pipeline state's *local transform* is set to the identity matrix. The *global transform hints* are set to *None* or to a suitable value for the new matrix and the *local transform hints* are set to *Identity*. A new member is added to the renderer's *current path* attribute with its *structure_id* set to *s_id* and its *offset* set to 0. When all the elements in the structure specified by *s_id* have been processed, control will be returned. Also the saved pipeline state attribute values are restored and the last member of the *current path* renderer attribute is deleted.

If the condition is not satisfied, processing of output commands will continue without invoking the structure specified by *s_id*.

If *s_id* references a non-existent structure or if the value given for *test* is not registered, an *OutputCommand* error is reported.

**Encoding: Conditional Execute Structure OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 135 | output command number |
| 2 | 4 | output command length |
| 4 | BITMASK | mask |
| 2 | CONDITION_TEST | condition flags test |
| 2 | | unused |
| 4 | STRUCTURE_ID | structure ID |

### 6.10.4 Conditional Include Structure

*mask*: BITMASK

*test*: CONDITION_TEST

*s_id*: STRUCTURE_ID

This command transfers flow-of-control to the specified structure if the condition is satisfied. The condition is satisfied if the bits of the pipeline state's *condition flags* corresponding to the bits set in *mask* meet the condition described by *test*. The registered values for *test* are shown in Table 6-53 (§6.10.3).

If the condition is satisfied, processing of output commands will commence with the first structure element in the structure specified by *s_id*. A new member is added to the renderer's *current path* attribute with its *structure_id* set to *s_id* and its *offset* set to 0. When all the elements in the structure specified by *s_id* have been processed, control will be returned and the last member of the *current path* renderer attribute will be deleted. Unlike output command Conditional Execute Structure, the renderer's pipeline state attribute values are not saved before the structure specified by *s_id* is invoked and restored when the processing of that structure has been completed.

If the condition is not satisfied, processing of output commands will continue without invoking the structure specified by *s_id*.

If *s_id* references a non-existent structure or if the value given for *test* is not registered, an *OutputCommand* error is reported.

**Encoding: Conditional Include Structure OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 136 | output command number |
| 2 | 4 | output command length |
| 4 | BITMASK | mask |
| 2 | CONDITION_TEST | condition flags test |
| 2 | | unused |
| 4 | STRUCTURE_ID | structure identifier |

### 6.10.5 Conditional Return

*mask*: BITMASK

*test*: CONDITION_TEST

If the condition is satisfied, this command returns flow-of-control to the parent structure and removes the latest member of the *current path* renderer attribute. If the condition is satisfied when the root structure, a **PEXRenderOutputCommands** request, or a **PEXRenderElements** request is being processed, all further output commands are ignored until the next **PEXEndStructure** or explicit or implicit **PEXEndRendering** request is encountered. If the condition is not satisfied, processing of output commands will continue.

The condition is satisfied if the bits of the pipeline state's *condition flags* corresponding to the bits set in *mask* meet the condition described by *test*. The registered values for *test* are shown in Table 6-53 (§6.10.3).

If the value given for *test* is not registered, an *OutputCommand* error is reported.

**Encoding: Conditional Return OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 137 | output command number |
| 2 | 3 | output command length |
| 4 | BITMASK | mask |
| 2 | CONDITION_TEST | condition flags test |
| 2 | | unused |

### 6.10.6 Execute Structure

5.0

*s_id*: STRUCTURE_ID

This command transfers flow-of-control to the structure specified by *s_id*. Processing of output commands will commence with the first structure element in the structure specified by *s_id*. All the pipeline state attribute values of the renderer are saved. The composite modeling transform is copied into the pipeline state's *global transform* and the pipeline state's *local transform* is set to the identity matrix. The *global transform hints* are set to *None* or to a suitable value for the new matrix and the *local transform hints* are set to *Identity*. A new member is added to the renderer's *current path* attribute with its *structure_id* set to *s_id* and its *offset* set to 0. When all the elements in the structure specified by *s_id* have been processed, control is returned, the saved pipeline state attribute values are restored, and the last member of the renderer's *current path* is deleted.

If *s_id* references a non-existent structure, an *OutputCommand* error is issued.

**Encoding: Execute Structure OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 75 | output command number |
| 2 | 2 | output command length |
| 4 | STRUCTURE_ID | structure identifier |

### 6.10.7 Include Structure

*s_id*: STRUCTURE_ID

This command transfers flow-of-control to the structure specified by *s_id*. Processing of output commands commences with the first structure element in that structure. A new member is added to the renderer’s *current path* attribute with its *structure_id* set to *s_id* and its *offset* set to 0. When all the elements in the structure specified by *s_id* have been processed, control is returned and the last member of the *current path* renderer attribute is deleted. Unlike output command Execute Structure, the renderer’s pipeline state attribute values are not saved before the structure specified by *s_id* is invoked and restored when the processing of that structure is completed. Neither are the pipeline state’s *global transform* or *local transform* modified.

If *s_id* references a non-existent structure, an *OutputCommand* error is issued.

**Encoding: Include Structure OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 138 | output command number |
| 2 | 2 | output command length |
| 4 | STRUCTURE_ID | structure identifier |

### 6.10.8 Label

5.0

*label*: INT32

When included in a **PEXStoreElements** request, this command inserts a structure element which is useful as a navigation aid during structure editing. This output command has no affect during rendering.

**Encoding: Label OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 76 | output command number |
| 2 | 2 | output command length |
| 4 | INT32 | label |

### 6.10.9 No-op

5.1

This output command has no affect during rendering. It is useful for client-side traversals accumulating state.

**Encoding: No-op OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 104 | output command number |
| 2 | 1 | output command length |

### 6.10.10 Pick ID

5.0

*pickid*: CARD32

This command sets the *pick id* entry of the renderer's pipeline state. The pick identifier active for a particular primitive is useful for navigating while picking since it is returned as part of the pick path.

Pick ID is a member of the Global attribute set.

**Encoding: Pick ID OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 68 | output command number |
| 2 | 2 | output command length |
| 4 | CARD32 | pick identifier |

### 6.10.11 Sequential Processing

*sequential_processing*: BOOLEAN

This command sets the *sequential processing* entry of the renderer's pipeline state.

Requests are executed in sequential order for a given client connection. Requests which process output commands are not guaranteed to be atomic with respect to concurrent rendering to the same destination drawable. This behavior is not modified by this command. Behavior which is affected is as follows:

If *sequential processing* is *True*, atomicity of execution is guaranteed for each individual OC, excluding Execute Structure, Include Structure, Conditional Execute Structure, Conditional Include Structure, GDP, GDP 2D, and GSE. Also, all requests and OCs will be executed in sequential order for a given connection.

If *False*, the renderer is released from the requirement to maintain strict atomic and sequential ordering of OCs. All attributes and transforms must be correctly bound to the primitives to which they apply.

An *OutputCommand* error is reported if the value in *sequential_processing* is not *True* or *False*.

**Encoding: Sequential Processing OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 123 | output command number |
| 2 | 2 | output command length |
| 1 | BOOLEAN | sequential processing |
| 3 | | unused |

### 6.10.12 Test Extent

| | |
|---|---|
| *corner1*: | COORD_3D |
| *corner2*: | COORD_3D |
| *index*: | TABLE_INDEX |
| *condition*: | RELATIONAL_TEST |
| *clip_test_mask*: | BITMASK |
| *size_test_mask*: | BITMASK |

This command modifies the *condition flags* entry of the renderer's pipeline state based on the results of a clip test and size test against the given extent ("bounding box").

The *corner1* and *corner2* positions define minimum and maximum *X*, *Y*, and *Z* coordinates in modeling coordinates. From these two corners, which define the diagonal corners of the extent, eight corners are specified. The eight corners are transformed to NPC space. For the purposes of this transformation, all model clipping is disabled. In NPC space, an axis-aligned volume that just bounds the transformed corners is determined.

The clip test is conducted first. The NPC space volume is clipped, then transformed to DC space. DC space clipping, defined by the *clip list* renderer attribute and window occlusion, is not conducted. If the extent is found to be completely outside the *NPC subvolume* or *viewport* (both renderer attributes), the *clip_test_mask* is OR'ed into the pipeline state's *condition flags* field. Otherwise the complement of the *clip_test_mask* is AND'ed into the pipeline state's *condition flags*.

After the results of the clip test have been used to modify the pipeline state's *condition flags*, the size test is conducted. The extent's *XY* DC space diagonal length is calculated in an implementation-dependent manner. However, it is conceptually the Euclidean distance defined by:

$$\sqrt{(x1-x0)^2 + (y1-y0)^2} \qquad \textbf{Equation 6.10}$$

The use of Euclidean distance may be replaced by a different measure, such as largest absolute value or average absolute value, if the result reasonably measures the extent's diagonal size in pixels.

If the extent is completely clipped, the extent size is automatically zero. If the extent is partially clipped or fully visible, the NPC volume representing the extent is again transformed to DC space. This time, however, NPC clipping is not performed, and as for the clip text, DC clipping is not done either. This results in the extent size representing the full size in DC space of the object, independent of how the object is clipped.

The extent's size is compared to the *extent size* defined in the entry, selected by the value in *index,* of the renderer's *extent size lookup table*. *condition* defines how the diagonal length is compared to the *extent size.* If the comparison is true, the *size_test_mask* is OR'ed into the pipeline state's *condition flags* field. Otherwise the complement of the *size_test_mask* is ANDed into the pipeline state's *condition flags.*

The extent tests are defined to err on the side of determining an object is more visible, and its size is slightly larger, than a more precise test would find. Implementations may choose to implement higher quality extent tests.

The *condition flags* are also modified with the Condition Flags OC (§6.10.2). The *condition flags* are used when processing the Conditional Execute Structure, Conditional Include Structure, and Conditional Return OCs.

An *OutputCommand* error is issued if *index* is zero or if *condition* is not one of the registered relational test values.

**Encoding: Test Extent OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 139 | output command number |
| 2 | 4+6**fp*/4 | output command length |
| 3**fp* | COORD_3D | corner 1 |
| 3**fp* | COORD_3D | corner 2 |
| 2 | TABLE_INDEX | extent size table index |
| 2 | RELATIONAL_TEST | relational test |
| 4 | BITMASK | extent clip test mask |
| 4 | BITMASK | extent size test mask |

### 6.10.13 Test Extent 2D

5.2

| | |
|---|---|
| *corner1*: | COORD_2D |
| *corner2*: | COORD_2D |
| *index*: | TABLE_INDEX |
| *condition*: | RELATIONAL_TEST |
| *clip_test_mask*: | BITMASK |
| *size_test_mask*: | BITMASK |

This command modifies the *condition flags* entry of the renderer's pipeline state based on the results of a clip test and size test against the given extent. The command is a 2D form of the Test Extent output command (§6.10.12), and differs from it only by containing a 2D value for *corner1* and *corner2*. The Z-coordinate of the corners is implicitly zero. Once these implicit values are associated with the output command, it is processed as a Test Extent output command.

**Encoding: Test Extent 2D OC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | 140 | output command number |
| 2 | 4+*fp* | output command length |
| 2*fp* | COORD_2D | corner 1 |
| 2*fp* | COORD_2D | corner 2 |
| 2 | TABLE_INDEX | extent size table index |
| 2 | RELATIONAL_TEST | relational test |
| 4 | BITMASK | extent clip test mask |
| 4 | BITMASK | extent size test mask |

# 7 Lookup Tables

## 7.1 Lookup Tables—General

A *lookup table* is a PEX resource that holds collections of related values. Lookup tables are sometimes used to support indirection for some output primitive attributes, but are also used for information such as light source descriptions, depth-cue parameters, and view definitions. Lookup tables consist of table entries, with each entry holding potentially different values for the set of parameters particular to the table type. Each entry in a table is addressed by an integer index, although whether the first allowable index is 0 or 1 is dependent on the type of table. The table descriptions in Section 7.2 give the entry definition for each table type.

Tables are explicitly created by the client, using the request **PEXCreateLookupTable** (§7.3.3). Entries are added or redefined with the **PEXChangeTableValues** request (§7.3.1) and deleted with the request **PEXDeleteTableEntries** (§7.3.4). Entries in a table need not have sequential indices. A table may contain entries 5, 7, and 10, for example, without containing entries 1 through 4, 6, 8, or 9. If a table has no entry for a particular index, then that entry is said to be undefined for that table. Initially, all entries not predefined (§7.1.3) are undefined.

The requests that create, manage, modify, and inquire a lookup table resource are listed in Table 7-1 .

**Table 7-1 . Lookup Table Resource Management Requests**

| Request | Description |
|---|---|
| **PEXChangeTableValues** | Define or set selected fields of an entry of a lookup table. |
| **PEXCreateLookupTable** | Create a lookup table resource. |
| **PEXCopyLookupTable** | Clear the destination lookup table, then copy the entries of the source lookup table into it. |
| **PEXDeleteTableEntries** | Remove selected entries from a lookup table. |
| **PEXFreeLookupTable** | Free a lookup table resource. |
| **PEXGetDefinedIndices** | Return the indices of the defined entries of a lookup table. |
| **PEXGetTableInfo** | Return the predefined entries and maximum number of entries of a lookup table. |
| **PEXGetTableValues** | Return selected fields of an entry of a lookup table. |

### 7.1.1 Fallback Entries and Values

Each lookup table has a designated fallback entry. This entry is used if an entry selected during rendering is undefined. If the fallback entry is needed but is also undefined, the fallback table values for each field of the table are used. The fallback entry and fallback values are listed with the definition of each table type in Section 7.2.

The behavior if the identifier for a referenced lookup table is Null is defined in §10.2. The fallback values are used if a valid table resource is referenced but the selected and fallback entries are undefined.

### 7.1.2 Table Values and Table Entries

Some lookup table requests have parameter *fields* of type TABLE_BITMASK and parameter *values* of type LISTofVALUE. *fields* indicates which values of the table entry are to be affected by the request. *values* holds those values. The values must be ordered in *values* proceeding from the least significant bit of *fields* to the most significant bit. Only those values indicated by *fields* are to be included in *values*. Each value encoding is padded to a multiple of 4 bytes. Values which could be encoded in less space are assumed to be placed in the least significant bits of the word.

TABLE_BITMASK, defined in §3.7.7.135, is the union of the bitmasks defined for each type of table. These bitmasks are defined with each table type in the following sections.

Some of the lookup table requests which are provided for compatibility with PEX versions previous to 5.2 use type TABLE_ENTRY. TABLE_ENTRY, defined in §3.7.7.136, is the union of the entries defined for each type of table. These entries are defined with table types defined prior to version 5.2 in the following sections. Fields added to a table at PEX version 5.2 or later are not included in the table entry type for the table.

### 7.1.3 Predefined Entries

A PEX implementation may predefine some entries in a lookup table when the table is created. Predefined entries are sequential and include the first legal entry of the table. The number of entries predefined is implementation-dependent, may be zero, and may vary for each table type. Predefined entries can be redefined by the client. The number of predefined entries for a given table type is returned by **PEXGetTableInfo**. The entries that a server predefines for a particular table type and sample drawable can be determined with **PEXGetPredefinedEntries**.

As of PEX 5.2, it is not recommended implementations provide predefined entries. The concept is explained here and supported to maintain compatibility with PEX versions previous to 5.2.

### 7.1.4 Set and Realized Values

Some table values set by the client may not be realizable. If not, then the value is mapped by PEX to the nearest value that is realizable. This mapped value is called the “realized” value and can be determined with the request **PEXGetTableValues**.

## 7.2 Table Types, Availability, and Definitions

### 7.2.1 Table Type Specification Syntax

When embedded in descriptions, table names are lower case with separating spaces (e.g., interior bundle table). Table entry fields are lower case, use separating spaces, and are printed in italics (e.g., *color type*).

Table types are described as follows:

**Type**

A brief definition of the table type’s usage.

A table which defines the fields of an entry of the table, including their names, encodings, and default values.

The functional description of the table type.

The situations under which an error arises.

The specification of the valid range of table indices and the fallback table entry.

The table entry encoding (if the table type was defined prior to PEX version 5.2).

### 7.2.2 Color Table

Color lookup tables are used to resolve indirect color references during rendering.

**Encoding: Color Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *color* | | | |
| | 2 | COLOR_TYPE | color type | RGBFloat |
| | 2 | | unused | |
| | *sc* | DIRECT_COLOR | color value (see COLOR for *sc*) | (1.0, 1.0, 1.0) |

†1) COLOR_TABLE_BITMASK : BITMASK

Color table entries hold a single direct-color value and the color type of that value. The color type need not be the same among color table entries. Only direct colors can be stored in color lookup tables; it is not possible to store a color of type *Indexed* (§3.7.6).

The color table is accessed during rendering whenever a color of type *Indexed* is encountered. The value of the *Indexed* color selects the desired entry of the color table.

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type or color type *Indexed* is given.

Valid color table indices are in the range [0, 65534]. The fallback entry is entry 1.

**Encoding: COLOR_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | COLOR_TYPE | color type |
| 2 | | unused |
| *sc* | DIRECT_COLOR | color value (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.3 Color Approximation Table 5.0

Color approximation lookup tables hold the information necessary to convert PEX colors to X11 colormap pixel values.

**Encoding: Color Approximation Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *approximation_method* 5.0 | | | |
| | 4 | COLOR_APPROX_TYPE | color approximation type | (imp-dep) |
| 0x00000002 | *color_model* 5.0 | | | |
| | 4 | COLOR_APPROX_MODEL | color approximation model | (imp-dep) |
| 0x00000004 | *max1* 5.0 | | | |
| | 4 | CARD16 | max1 | (imp-dep) |
| 0x00000008 | *max2* 5.0 | | | |
| | 4 | CARD16 | max2 | (imp-dep) |
| 0x00000010 | *max3* 5.0 | | | |
| | 4 | CARD16 | max3 | (imp-dep) |
| 0x00000020 | *dither* 5.0 | | | |
| | 4 | SWITCH | dither | (imp-dep) |
| 0x00000040 | *mult1* 5.0 | | | |
| | 4 | CARD32 | mult1 | (imp-dep) |
| 0x00000080 | *mult2* 5.0 | | | |
| | 4 | CARD32 | mult2 | (imp-dep) |
| 0x00000100 | *mult3* 5.0 | | | |
| | 4 | CARD32 | mult3 | (imp-dep) |
| 0x00000200 | *weight1* 5.0 | | | |
| | *fp* | FLOAT | weight1 | (imp-dep) |
| 0x00000400 | *weight2* 5.0 | | | |
| | *fp* | FLOAT | weight2 | (imp-dep) |
| 0x00000800 | *weight3* 5.0 | | | |
| | *fp* | FLOAT | weight3 | (imp-dep) |
| 0x00001000 | *base_pixel* 5.0 | | | |
| | *4* | CARD32 | base pixel | (imp-dep) |

†1) COLOR_APPROX_TABLE_BITMASK : BITMASK

Color approximation converts colors computed by PEX to pixel values. The PEX rendering pipeline produces a color in a multi-dimensional color space such as RGB. That color—called the input color—is converted by PEX color approximation to a pixel value that selects an entry in an X11 colormap. The method and parameters used to convert the color are taken from the color approximation table entry indicated by the pipeline state’s *color approximation index* (§6.9.24). Alpha values are unaffected by color approximation.

During color approximation, each color produced by the rendering pipeline is first converted to the color model specified in the selected color approximation table entry. (When that color model is the same as the rendering color model, this is an identity mapping.) The color approximation color models defined by PEX are listed in Table 7-2 .

**Table 7-2 . Enumerated Type *ColorApproxModel*[†1]**

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *RGB* | red, green, blue color model 5.0 |
| 2 | *CIE* | CIELUV color model 5.0 |
| 3 | *HSV* | hue, saturation, value 5.0 |
| 4 | *HLS* | hue, lightness, saturation 5.0 |
| 5 | *YIQ* | NTSC luminance (Y), inphase (wideband orange-cyan), quadrature (narrowband magenta-green) 5.0 |

†1) See §3.7.6 for a description of the color models listed here except the YIQ model.

After the input color is converted to the color approximation color model, the converted color is converted to a pixel value using the method indicated by the *color approximation type* in the selected table entry. PEX defines two color approximation types, listed in Table 7-3 .

**Table 7-3 . Enumerated Type *ColorApproxType***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *ColorSpace* | Intended to be used with *DirectColor*, *TrueColor*, *PseudoColor*, and *StaticColor* visuals. 5.0 |
| 2 | *ColorRange* | Intended to be used with *GrayScale* and *StaticGray* visuals. 5.0 |

### *ColorSpace* Color Approximation

In *ColorSpace* color approximation, each component of a color, $(C_1, C_2, C_3)$, is scaled by the corresponding "max" value, *(max1, max2, max3)*, in the selected color approximation table entry, as follows:

$$C_1 = max1 \cdot C_1, \quad C_2 = max2 \cdot C_2, \quad C_3 = max3 \cdot C_3 \qquad \textbf{Equation 7.1}$$

The $C_i$ are conceptually rounded to integers in this operation.

*max1*, *max2*, and *max3* typically indicate the number of samplings (minus 1) of each color component in a colormap organized as a color cube. For example, the *max* values in a colormap organized as a 3-3-2 color cube would be 7, 7, 3 ($2^3$–1, $2^3$–1, $2^2$–1). Equation 7.1 thus serves to scale the color coordinates into the available range of discrete colors.

The $C_i$ are converted to a pixel value by the formula given in Equation 7.2[†1]:

$$P = C_1 \cdot mult1 + C_2 \cdot mult2 + C_3 \cdot mult3 + base\ pixel \qquad \textbf{Equation 7.2}$$

Where *mult1*, *mult2*, *mult3*, and *base pixel* are taken from the color approximation entry, and *P* represents the resulting pixel value written to the drawable. *mult1, mult2*, and *mult3* typically are used to effectively shift the color components to the bits within the pixel representing that color component. For example, the *mult* values in a colormap organized as

†1) The use of addition rather than logical OR for composing pixel values permits allocations where the primary components are not allocated into distinct bitplanes.

a 3-3-2 color cube, where bits 5 - 7 of the pixel represent the $C_1$ component, bits 2 - 4 the $C_2$ component, and bits 0 and 1 the $C_3$, component, would be $2^5$, $2^2$, and 1. The *base_pixel* is used to shift the values up to the first pixel of the color cube within the colormap.

In *ColorSpace* color approximation, the *weight1*, *weight2*, and *weight3* values in the color approximation entry are not used.

### *ColorRange* Color Approximation

In *ColorRange* approximation, the color components in the color approximation color model, $(C_1, C_2, C_3)$, are multiplied by the corresponding weights, (*weight1*, *weight2*, *weight3*), taken from the color approximation entry. These terms are added together to form a single, conceptual floating-point value:

$$V = C_1 \cdot weight1 + C_2 \cdot weight2 + C_3 \cdot weight3$$ **Equation 7.3**

The weights may emphasize, minimize, or eliminate one or more of the color components. For example, weights of (0.299R, 0.587G, 0.114B) convert an RGB value to a single-valued intensity used in the NTSC color standard. The weights typically should be normalized such that they sum to 1; PEX does not normalize the weights.

After the weights are applied, the computed value, *V*, is multiplied by *max1* in the color approximation entry:

$$V' = V \cdot max1$$ **Equation 7.4**

*max2* and *max3* are not used.

The value *V'* is replicated for each of the three color components. Each component is then multiplied by *mult1*, *mult2*, or *mult3*, respectively, and the values added together with *base pixel* to form the pixel value, *P*, written to the drawable:†2

$$P = V' \cdot mult1 + V' \cdot mult2 + V' \cdot mult3 + base\ pixel$$ **Equation 7.5**

*max1* is typically one less than the number of entries in a colormap. For example, a value of 99 for *max1* would scale colors to the range [0, 99], and might be used to display an image in gray scale using a colormap with 100 contiguous shades of gray. *Typically mult1 is 1 and mult2 and mult3 are 0. The ability to specify mult2 and mult3 is supported so gray ramps can be simulated on a DirectColor visual by replicating the intensity in all three channels of the pixel.*

### Multipliers

Inquire *BestColorApproxValues* with request **PEXGetImpDepConstants** to determine what values, if any, of *mult1, mult2,* and *mult3* are likely to result in the best performance.

### Dithering

*dither* is a hint to the renderer as to whether some attempt at dithering should be performed. With request **PEXGetImpDepConstants** inquire *DitheringSupported* to determine if the server can dither, and inquire *DitheringControllable* to determine if dithering can be disabled with the dithering hint, given the server does dither. If dithering is supported, the algorithm used is implementation-dependent.

The color approximation table is always accessed when rendering. The value of the pipeline state's *color approximation index* selects the desired entry of the table.

A *Value* error is issued if an unsupported color approximation type or color approximation model is given.

†2) The use of addition rather than logical OR for composing pixel values permits allocations where the primary components are not allocated into distinct bitplanes.

Valid color approximation lookup table indices are in the range [0, 65534]. The fallback entry is entry 0.

**Encoding: COLOR_APPROX_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | COLOR_APPROX_TYPE | color approximation type |
| 2 | COLOR_APPROX_MODEL | color approximation model |
| 2 | CARD16 | max1 |
| 2 | CARD16 | max2 |
| 2 | CARD16 | max3 |
| 1 | SWITCH | dither |
| 1 | | unused |
| 4 | CARD32 | mult1 |
| 4 | CARD32 | mult2 |
| 4 | CARD32 | mult3 |
| *fp* | FLOAT | weight1 |
| *fp* | FLOAT | weight2 |
| *fp* | FLOAT | weight3 |
| 4 | CARD32 | base pixel |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.4 Data Mapping Table

Data mapping lookup tables are used to define how color or other application-specific data is mapped to determine the intrinsic color of fill area output primitives with extended data.

**Encoding: Data Map Table Values**

| Bitmask[†1] | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *method* | | | |
| | 4 | DATA_MAP_METHOD | data mapping method | *None* |
| 0x00000002 | *source_selectors* | | | |
| | 2 | CARD16 | number of source selectors (*n*) | 3 |
| | 2 | | unused | |
| | 2*n | LISTofCARD16 | prioritized list of source selectors | {*VertexColor, FacetColor, SurfaceColor*} |
| | | 0 *SurfaceColor* | | |
| | | 1 *VertexColor* | | |
| | | 2 *FacetColor* | | |
| | *p* | | *p*=pad(2**n*) | |

†1) DATA_MAP_TABLE_BITMASK : BITMASK

Data mapping is the first stage in the color pipeline.

See §6.9.27 for the defined values of enumerated type *DataMappingMethod.*

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on data mapping lookup tables (§7.4).

The data mapping table is accessed during rendering whenever the pipeline state’s *data map method ASF* or *bf data map method ASF* entry is *Bundled*. The values of the pipeline state’s *data map index* and *bf data map index* select the desired entries of the table.

A *Value* error is issued if an unsupported data mapping method is given, if the data record size doesn’t match the method, or if the method is *Color* and an undefined source selector is given.

Valid data mapping table indices are in the range [1, 65535]. The fallback entry is entry 1.

### 7.2.5 Depth Cue Table 5.0

Depth cueing lookup tables are used to blend color computed by the lighting, shading, and texturing stages of the rendering pipeline with a depth cue color. The blending is controlled relative to the depth in NPC space. Often depth cueing is used to blend the portion of the image furthest away from the viewer with the background color, heightening the sensation of depth in the image. Color alpha components are not depth cued.

**Encoding: Depth Cue Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *depth_cue_mode* 5.0 | | | |
| | 4 | SWITCH | depth cue mode | *Off* |
| 0x00000002 | *front_reference_plane* 5.0 | | | |
| | *fp* | FLOAT | front reference plane | 1.0 |
| 0x00000004 | *back_reference_plane* 5.0 | | | |
| | *fp* | FLOAT | back reference plane | 0.0 |
| 0x00000008 | *front_scale_factor* 5.0 | | | |
| | *fp* | FLOAT | front scale factor | 1.0 |
| 0x00000010 | *back_scale_factor* 5.0 | | | |
| | *fp* | FLOAT | back scale factor | 0.0 |
| 0x00000020 | *depth_cue_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | depth cue color (see COLOR for *sc*) | (RGBFloat, (0.0, 0.0, 0.0)) |

†1) DEPTH_CUE_TABLE_BITMASK : BITMASK

Depth cueing is performed if the value of the *depth cue mode* parameter in the selected depth cue table entry is *On*. It is not performed if the value of that parameter is *Off*.

Primitives are depth cued by blending the reflected and textured color of its points with the specified *depth cue color*. The front and back reference planes are perpendicular to the NPC Z axis, and are specified by the $Z_{NPC}$ coordinate at which they intersect that axis. The scale factors must be in the range [0.0, 1.0].

The depth cueing table is always accessed when rendering. The value of the pipeline state's *depth cueing index* selects the desired entry of the table.

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an undefined depth cue mode is given, if the *back reference plane* is greater than the *front reference plane*, or if the *front scale factor* or *back scale factor* is not in the range [0.0, 1.0].

Valid depth cue table indices are in the range [0, 65534]. The fallback entry is entry 0.

**Encoding: DEPTH_CUE_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | SWITCH | depth cue mode |
| 3 | | unused |
| *fp* | FLOAT | front reference plane |
| *fp* | FLOAT | back reference plane |
| *fp* | FLOAT | front scale factor |
| *fp* | FLOAT | back scale factor |
| 4+*sc* | COLOR_SPECIFIER | depth cue color (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.6 Edge Bundle Table 5.0

Edge bundle lookup tables contain collections of edge attribute values.

**Encoding: Edge Bundle Table Values**

| **Bitmask**†1 | **# of Bytes** | **Data Type or Value** | **Description** | **Fallback** |
|---|---|---|---|---|
| 0x00000001 | *surface_edge_flag* 5.0 | | | |
| | 4 | SWITCH | edge flag (§6.9.81) | *Off* |
| 0x00000002 | *surface_edge_type* 5.0 | | | |
| | 4 | SURFACE_EDGE_TYPE | edge type (§6.9.82) | *Solid* |
| 0x00000004 | *surface_edge_width* 5.0 | | | |
| | *fp* | FLOAT | edge width (§6.9.83) | 1.0 |
| 0x00000008 | *surface_edge_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | edge color (see COLOR for *sc*) (§6.9.79) | (*Indexed*, 1) |

†1) EDGE_TABLE_BITMASK : BITMASK

The edge bundle table is accessed during rendering whenever the aspect source flag of one of the edge attributes stored in the edge bundle table is *Bundled* (§2.1.4.1). The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *edge bundle index* selects the desired entry of the table. The value of the bundled edge attributes in that entry are then used to set the values of those edge attributes.

For more information about each edge attribute, see the corresponding edge attribute output command (the section numbers of the edge bundle table values are given with the field name in the encoding above). (These output commands set pipeline state which is used whenever the aspect source flag for an edge attribute is *Individual* rather than *Bundled.*)

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if the surface edge flag is not *On* or *Off*, or if an unsupported edge type is given.

Valid edge bundle lookup table indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: EDGE_TABLE_ENTRY**†1

| **# of Bytes** | **Data Type or Value** | **Description** |
|---|---|---|
| 1 | SWITCH | edge flag |
| 1 | | unused |
| 2 | SURFACE_EDGE_TYPE | edge type |
| *fp* | FLOAT | edge width |
| 4+*sc* | COLOR_SPECIFIER | edge color (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.7 Extent Size Table

Extent size lookup tables hold the extent sizes used with Test Extent and Test Extent 2D OCs (§6.10.13 and §6.10.12). These output commands compare the length of the diagonal of an extent volume or rectangle with a extent size obtained from the table. The results of this comparison determine modifications to the pipeline state's *condition flags* which are used to control the behavior of subsequent Conditional Execute Structure (§6.10.3), Conditional Include Structure (§6.10.4), and Conditional Return (§6.10.5) OCs.

**Encoding: Extent Size Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *extent_size* | | | |
| | 4 | CARD32 | extent size (DCs) | 0 |

†1) EXTENT_SIZE_TABLE_BITMASK : BITMASK

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on extent size lookup tables (§7.4).

The extent size table is accessed during rendering whenever a Test Extent or Test Extent 2D output command is encountered. The value of the output command's *index* parameter selects the desired entry of the table.

Valid extent size table indices are in the range [1, 65535]. The fallback entry is entry 1.

### 7.2.8 Highlight Table

Highlight table entries indicate how primitives are highlighted.

**Encoding: Highlight Table Values**

| **Bitmask**†1 | **# of Bytes** | **Data Type or Value** | **Description** | **Fallback** |
|---|---|---|---|---|
| 0x00000001 | *highlight_method* | | | |
| | 4 | HIGHLIGHT_METHOD | highlight method | 1 (imp-dep) |
| 0x00000002 | *highlight_color* | | | |
| | 4+*sc* | COLOR_SPECIFIER | highlight color (see COLOR for *sc*) | (imp-dep) |

†1) HIGHLIGHT_TABLE_BITMASK : BITMASK

The registered highlight methods are listed in Table 7-4 . Each method indicates if and which other values of the highlight table entry are used, and in what way.

**Table 7-4 . Enumerated Type *HighlightMethod***

| **Value** | **Mnemonic** | **Description** |
|---|---|---|
| 1 | (implementation-dependent) | Highlight a primitive in an implementation-dependent fashion. |
| 2 | *Color* | Highlight a primitive by overriding its color with the color defined by field *highlight_color* of the active highlight table entry. Lighting, shading, depth cueing, transparency, texture mapping, and data mapping are not applied to the primitive. For surface primitives, the highlight color is applied to both the front-facing and back-facing facets. |

Highlighting is removed by clearing and rerendering the image.

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on highlight lookup tables (§7.4).

Highlighting is performed when a primitive is rendered and the pipeline state's *name set* is contained in the renderer's *highlight inclusion* name set and not in the renderer's *highlight exclusion* name set. The value of the pipeline state's *highlight index* selects the desired entry of the table.

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an unsupported highlight method is given.

Valid highlight table indices are in the range [1, 65535]. The fallback entry is entry 1.

### 7.2.9 Interior Bundle Table 5.0

**Encoding: Interior Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *interior_style* 5.0 | | | |
| | 4 | INTERIOR_STYLE | interior style (§6.9.84) | *Hollow* |
| 0x00000002 | *interior_style_index* 5.0 | | | |
| | 4 | TYPE_OR_TABLE_INDEX | interior style index (§6.9.85) | 1 |
| 0x00000004 | *reflection_model* 5.0 | | | |
| | 4 | REFLECTION_MODEL | reflection model (§6.9.88) | *None* |
| 0x00000008 | *interp_method* 5.0 | | | |
| | 4 | SURFACE_INTERP | surface interpolation method (§6.9.86) | *None* |
| 0x00000010 | *bf_interior_style* 5.0 | | | |
| | 4 | INTERIOR_STYLE | backface interior style (§6.9.15) | *Hollow* |
| 0x00000020 | *bf_interior_style_index* 5.0 | | | |
| | 4 | TYPE_OR_TABLE_INDEX | backface interior style index (§6.9.16) | 1 |
| 0x00000040 | *bf_reflection_model* 5.0 | | | |
| | 4 | REFLECTION_MODEL | backface reflection model (§6.9.19) | *None* |
| 0x00000080 | *bf_interp_method* 5.0 | | | |
| | 4 | SURFACE_INTERP | backface surface interpolation method (§6.9.17) | *None* |
| 0x00000100 | *surface_approx* 5.0 | | | |
| | 4+2*fp* | SURFACE_APPROX | surface approximation (§6.9.75) | (1 (imp-dep), n/a, n/a) |
| 0x00000200 | *surface_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | surface color (see COLOR for *sc*) (§6.9.76) | (*Indexed*, 1) |
| 0x00000400 | *reflection_attributes* 5.0 | | | |
| | 4+*sc*+5*fp* | REFLECTION_ATTR | reflection attributes (see COLOR for *sc)* (§6.9.87) | (1.0, 1.0, 1.0, 0.0, 0,0, (*Indexed*, 1) |
| 0x00000800 | *bf_surface_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | backface surface color (see COLOR for *sc*) (§6.9.13) | (*Indexed*, 1) |
| 0x00001000 | *bf_reflection_attributes* 5.0 | | | |
| | 4+*sc*+5*fp* | REFLECTION_ATTR | backface reflection attributes (see COLOR for *sc*) (§6.9.18) | (1.0, 1.0, 1.0, 0.0, 0.0, (*Indexed*, 1) |
| 0x00002000 | *parametric_surface_char*†2 5.2 | | | |
| | 4+*n*+*p* | PSURF_CHAR | parametric surface characteristics (see PSURF_CHAR for *n* and *p*) (§6.9.63) | (*None*, n/a) |

†1) INTERIOR_TABLE_BITMASK : BITMASK
†2) The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the parametric surface characteristics value of an interior table entry (§7.4).

Interior bundle lookup tables hold collections of interior attribute values.

If the renderer's *PHIGS interior bundle mode* is *False*, the interior bundle table is accessed during rendering whenever the aspect source flag of one of the interior attributes stored in the interior bundle table is *Bundled* (§2.1.4.1). The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *interior bundle index* selects the desired entry of the table. The pipeline state's *backface interior bundle index, reflectance bundle index, backface reflectance bundle index,* and *parametric surface bundle index* are ignored. The value of the bundled interior attributes in that entry are then used to set the values of those interior attributes.

When the renderer's *PHIGS interior bundle mode* is *True*, the interior bundle is still accessed during rendering whenever the aspect source flag of one of the interior attributes stored in the interior bundle table is *Bundled*. However which pipeline state index is used to select the desired entry of the table depends on which interior attribute is bundled. Table 7-5 defines which pipeline index is used for each interior attribute in the Interior Bundle Table.

**Table 7-5 . Indices Used to Select Bundled Interior Attributes in the PHIGS Interior Bundle Mode**

| | |
|---|---|
| *interior bundle index* | Surface Interior Style |
| | Surface Interior Style Index |
| | Surface Color |
| | Surface Interpolation Method |
| *backface interior bundle index* | Backface Surface Interior Style |
| | Backface Surface Interior Style Index |
| | Backface Surface Color |
| | Backface Surface Interpolation Method |
| *reflectance bundle index* | Surface Reflection Model |
| | Surface Reflection Attributes |
| *backface reflectance bundle index* | Backface Surface Reflection Model |
| | Backface Surface Reflection Attributes |
| *parametric surface bundle index* | Surface Approximation |
| | Parametric Surface Characteristics |

For more information about each surface interior attribute, see the corresponding surface interior attribute output command (the section numbers of the interior table values are given with the field name in the encoding above). (These output commands set pipeline state which is used whenever the aspect source flag for a surface interior attribute is *Individual* rather than *Bundled.*)

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an unsupported interior style, backface interior style, reflection model, backface reflection model, surface interpolation method, backface surface interpolation method, surface approximation method, or parametric surface characteristics type is given, if a chordal size or chordal deviation surface approximation method tolerance is non-positive, if a relative surface approximation method tolerance is outside of the range [0.0, 1.0], if the parametric surface characteristics data record size doesn't match the type, if an isoparametric curve parametric surface characterization placement value is unsupported, or if a level curves parametric surface characterization direction vector has a zero length.

Valid interior bundle lookup table indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: INTERIOR_TABLE_ENTRY†1**

| # of Bytes | Data Type of Value | Description |
|---|---|---|
| 2 | INTERIOR_STYLE | interior style |
| 2 | TYPE_OR_TABLE_INDEX | interior style index |
| 2 | REFLECTION_MODEL | reflection model |
| 2 | SURFACE_INTERP | surface interpolation method |
| 2 | INTERIOR_STYLE | backface interior style |
| 2 | TYPE_OR_TABLE_INDEX | backface interior style index |
| 2 | REFLECTION_MODEL | backface reflection model |
| 2 | SURFACE_INTERP | backface surface interpolation method |
| 4+2**fp* | SURFACE_APPROX | surface approximation |
| 4+*sc* | COLOR_SPECIFIER | surface color (see COLOR for *sc*) |
| 4+*sc*+5**fp* | REFLECTION_ATTR | reflection attributes |
| 4+*sc* | COLOR_SPECIFIER | backface surface color (see COLOR for *sc*) |
| 4+*sc*+5**fp* | REFLECTION_ATTR | backface reflection attributes |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.10 Light Table 5.0

Each entry of a light lookup table holds a light source definition.

**Encoding: Light Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *light_type* 5.0 | | | |
| | 4 | LIGHT_TYPE | light type | (imp-dep) |
| 0x00000002 | *direction* 5.0 | | | |
| | 3**fp* | VECTOR_3D | direction vector | (imp-dep) |
| 0x00000004 | *position* 5.0 | | | |
| | 3**fp* | COORD_3D | position | (imp-dep) |
| 0x00000008 | *concentration* 5.0 | | | |
| | *fp* | FLOAT | concentration | (imp-dep) |
| 0x00000010 | *spread_angle* 5.0 | | | |
| | *fp* | FLOAT | spread angle | (imp-dep) |
| 0x00000020 | *constant_atten_factor* 5.0 | | | |
| | *fp* | FLOAT | constant attenuation factor | (imp-dep) |
| 0x00000040 | *linear_atten_factor* 5.0 | | | |
| | *fp* | FLOAT | linear attenuation factor | (imp-dep) |
| 0x00000080 | *light_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | light color (see COLOR for *sc*) | (imp-dep) |
| 0x00000100 | *concentration_clamp_region*†2 5.2 | | | |
| | *fp* | FLOAT | concentration clamp region | (imp-dep) |
| 0x00000200 | *quadratic_atten_factor*†2 5.2 | | | |
| | *fp* | FLOAT | quadratic attenuation factor | (imp-dep) |

†1) LIGHT_TABLE_BITMASK : BITMASK
†2) The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the concentration clamp region or quadratic attenuation factor values of a light table entry (§7.4).

The defined light types are listed in Table 7-6 .

**Table 7-6 . Enumerated Type *LightType***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Ambient* | Illuminates primitives uniformly independent of location and orientation. 5.0 |
| 2 | *WCS_Vector* | Simulates light emanating from infinity along a particular direction. 5.0 |
| 3 | *WCS_Point* | Simulates light emanating in all directions from a particular position. Control is provided over the rate at which the intensity drops off as a function of the distance from the light source. 5.0 |
| 4 | *WCS_Spot* | Simulates a spot light. This light emanates from a particular position within a cone of influence centered along a particular direction. Control is provided over the rate at which the intensity drops off both as a function of the distance from the light source and the distance from the center of the cone of influence. Additional control is provided over how sharply the intensity drops off along a region against the edge of the cone of influence. 5.0 |

*direction:* The vector (in WCs) along which the light travels from its source. Applies to the definition of light types *WCS_Vector* and *WCS_Spot*.

*position*: The location (in WCs) from which the light emanates. Applies to the definition of light types *WCS_Point* and *WCS_Spot.*

*concentration*: The factor used to define the rate at which a light’s intensity drops off as a function of the distance from the center of the cone of influence to the edge of the cone of influence. Applies to the definition of light type *WCS_Spot.*

*spread_angle*: The angle (in radians) from the light source direction vector which defines the cone of influence of the light. The light has no affect on points which lie outside the cone of influence. Applies to the definition of light type *WCS_Spot.*

*constant_atten_factor*: The constant factor used in the determination of the rate at which the light’s intensity falls off as a function of the distance from the light’s position. Applies to the definition of light types *WCS_Point* and *WCS_Spot.*

*linear_atten_factor*: The linear factor used in the determination of the rate at which the light’s intensity falls off as a function of the distance from the light’s position. Applies to the definition of light types *WCS_Point* and *WCS_Spot.*

*light_color:* The light’s color. Applies to the definition of all types of lights.

*concentration_clamp_region:* The angle (in radians) inward from the cone of influence edge which defines a region along the outer portion of the cone influence. In this region the concentration is multiplied by a linear ramp which drops from 1.0 at the edge of the region to 0.0 at the edge of the cone of influence. This has the affect of clamping the light’s intensity sharply down to zero at the edge of the cone of influence.
Inquire *ConcentrationClampSupported* with request **PEXGetImpDepConstants** to determine whether the concentration clamp region is actually applied in the reflectance calculation.
Applies to the definition of light type WCS_Spot.

*quadratic_atten_factor:* The quadratic factor used in the determination of the rate at which the light’s intensity falls off as a function of the distance from the light’s position.
Inquire *QuadraticAttenSupported* with request **PEXGetImpDepConstants** to determine whether the quadratic attenuation factor is actually used in the reflectance calculation.
Applies to the definition of light types *WCS_Point* and *WCS_Spot.*

Note all of the light table fields are not needed by each type of light. Fields not needed for the definition of a particular light are ignored.

The reflectance equations are implementation -dependent. However suggested reflectance equations are given in §2.2.2.2.

The light table is accessed during rendering whenever the pipeline state’s *light source state* is non-empty. The *light source state* selects entries of the light table which then represent the enabled lights.

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an unsupported light type is given or if a direction vector of length zero is given.

Valid light table indices are in the range [1, 65535]. There is not a fallback entry for the light table.

**Encoding: LIGHT_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | LIGHT_TYPE | light type |
| 2 | | unused |
| 3**fp* | VECTOR_3D | direction vector |
| 3**fp* | COORD_3D | position |
| *fp* | FLOAT | concentration |
| *fp* | FLOAT | spread angle |
| *fp* | FLOAT | constant attenuation factor |
| *fp* | FLOAT | linear attenuation factor |
| 4+*sc* | COLOR_SPECIFIER | light color (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.11 Line Bundle Table 5.0

Line bundle lookup tables contain collections of line attribute values.

**Encoding: Line Bundle Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *line_type* 5.0 | | | |
| | 4 | LINE_TYPE | line type (§6.9.48) | *Solid* |
| 0x00000002 | *line_interp_method* 5.0 | | | |
| | 4 | POLYLINE_INTERP | interpolation method (§6.9.68) | *None* |
| 0x00000004 | *curve_approx* 5.0 | | | |
| | 4+*fp* | CURVE_APPROX | curve approximation (§6.9.25) | (1 (imp-dep), n/a) |
| 0x00000008 | *line_width* 5.0 | | | |
| | fp | FLOAT | line width scale factor (§6.9.49) | 1.0 |
| 0x00000010 | *line_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | line color (see COLOR for *sc*) (§6.9.44) | (*Indexed*, 1) |
| 0x00000020 | *line_cap_style*†2 5.2 | | | |
| | 4 | LINE_CAP_STYLE | wide line cap style (§6.9.43) | Butt |
| 0x00000040 | *line_join_style*†2 5.2 | | | |
| | 4 | LINE_JOIN_STYLE | wide line join style (§6.9.46) | 0 (imp-dep) |

†1) LINE_TABLE_BITMASK : BITMASK

†2) The requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on the line cap style or join style values of a line bundle entry (§7.4).

The line bundle table is accessed during rendering whenever the aspect source flag of one of the line attributes stored in the line bundle table is *Bundled* (§2.1.4.1). The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state’s *line bundle index* selects the desired entry of the table. The value of the bundled line attributes in that entry are then used to set the values of those line attributes.

For more information about each line attribute, see the corresponding line attribute output command (the section numbers of the line bundle table values are given with the field name in the encoding above). (These output commands set pipeline state which is used whenever the aspect source flag for a line attribute is *Individual* rather than *Bundled.*)

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an unsupported line type, interpolation method, curve approximation method, cap style, or join style is given, if a chordal size or chordal deviation curve approximation method tolerance is non-positive, or if a relative curve approximation method tolerance is outside of the range [0.0, 1.0].

Valid line bundle indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: LINE_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | LINE_TYPE | line type |
| 2 | POLYLINE_INTERP | polyline interpolation method |
| 4+*fp* | CURVE_APPROX | curve approximation |
| *fp* | FLOAT | line width scale factor |
| 4+*sc* | COLOR_SPECIFIER | line color (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.12 Marker Bundle Table

Marker bundle lookup tables contain collections of marker attribute values.

**Encoding: Marker Bundle Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *marker_type* | | | |
| | 4 | MARKER_TYPE | marker type (§ 6.9.58) | Asterisk |
| 0x00000002 | *marker_scale_factor* | | | |
| | *fp* | FLOAT | marker scale factor (§ 6.9.57) | 1.0 |
| 0x00000004 | *marker_color* | | | |
| | 4+*sc* | COLOR_SPECIFIER | marker color (see COLOR for *sc*) (§ 6.9.55) | (*Indexed*, 1) |

†1) MARKER_TABLE_BITMASK : BITMASK

The marker bundle table is accessed during rendering whenever the aspect source flag of one of the marker attributes stored in the marker bundle table is *Bundled* (§ 2.1.4.1). The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state’s *marker bundle index* selects the desired entry of the table. The value of the bundled marker attributes in that entry are then used to set the values of those marker attributes.

For more information about each marker attribute, see the corresponding marker attribute output command (the section numbers of the marker bundle table values are given with the field name in the encoding above). (These output commands set pipeline state which is used whenever the aspect source flag for a marker attribute is *Individual* rather than *Bundled.*)

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535.

Valid marker bundle indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: MARKER_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | MARKER_TYPE | marker type |
| 2 | | unused |
| *fp* | FLOAT | marker scale factor |
| 4+*sc* | COLOR_SPECIFIER | marker color (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.13 Marker Glyph Table

Each entry of a marker glyph table defines a marker type.

**Encoding: Marker Glyph Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *marker_glyph* | | | |
| | 2 | ENUM_TYPE_INDEX | marker glyph type (see Table 7-7 ) | *Asterisk*†2 |
| | 2 | CARD16 | size of data (*n*) | n/a |
| | n | | marker glyph data†3 | n/a |
| | p | | *p*=pad(*n*) | n/a |

†1) MARKER_GLYPH_TABLE_BITMASK : BITMASK
†2) Registered marker type *Asterisk* of enumerated type *MarkerType*.
†3) The encoding of the marker glyph data depends on the marker glyph type. See below.

The marker glyph table is accessed whenever a marker is rendered. The value of the pipeline state's *marker type* selects the desired entry of the table.

Each entry of a marker glyph table consists of a marker glyph type and the data for that glyph. The registered marker glyph types are shown in Table 7-7 .

**Table 7-7 . Enumerated Type *MarkerGlyphType***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Stroke* | The marker glyph is defined as a list of 2D polylines. (0.0, 0.0) is mapped to the marker position transformed into DC space and the range [-1.0, 1.0] in each direction is mapped to the *NominalMarkerSize* times the pipeline state's *marker scale factor*, or the nearest supported size.<br>A single-coordinate polyline is rendered as a point. |

Marker glyph type = *Stroke* data :
[points : LISTofLISTofCOORD_2D]

**Encoding: Data for Marker Glyph Type *Stroke***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of polylines |
| *m*†1 | LISTofLISTofCOORD_2D | points |

†1) The polylines are sequentially defined. For each polyline, an initial CARD32 value defines the number of coordinates (*c*). The *c* coordinates, of type COORD_2D, follow.

A *Value* error is issued if an unsupported marker glyph type is given or if the data for a glyph of type *Stroke* contains a polyline with no coordinates.

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** (§7.4) do not operate on marker glyph tables.

Valid marker glyph table indices are in the range [1, 65535]. The fallback entry is entry 1.

### 7.2.14 Pattern Table 5.0

Each entry of a pattern lookup table holds a pattern definition.

**Encoding: Pattern Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *pattern_data* 5.0 | | | |
| | 2 | COLOR_TYPE | color type | (imp-dep) |
| | 2 | CARD16 | number of columns (*m*) | (imp-dep) |
| | 2 | CARD16 | number of rows (*n*) | (imp-dep) |
| | 2 | | unused | (imp-dep) |
| | *m*n*sc*†2 | LISTofCOLOR | cell colors (see COLOR for *sc*) | (imp-dep) |

†1) PATTERN_TABLE_BITMASK : BITMASK
†2) The 2D array is flat with the first row of colors followed by the second row of colors and so on.

The pattern is a grid of equally sized cells. Each cell has a single color. There are *m* x *n* cells. When applying interior style *Pattern* to a fill area primitive, the pattern reference point and vectors are projected along a normal vector onto the plane of each facet of the primitive. A parallelogram is formed with its origin defined by the projected reference point. One side of the parallelogram is defined by a vector whose direction is parallel to the first projected reference vector and whose length is equal to the absolute value of the X-component of the pattern size. The adjacent side of the parallelogram is defined similarly by the second projected reference vector and the absolute value of the Y-component of the pattern size. The pattern is applied across the parallelogram such that the first cell has the parallelogram origin in one corner. The first *m* colors in the pattern correspond to the row of cells nearest to and along the first projected reference vector. The pattern is mapped onto the facet by replicating the pattern parallelogram in directions parallel to the parallelogram's sides until the entire facet is covered.

The pattern table is accessed during rendering whenever the pipeline state's *interior style* or *bf interior style* is *Pattern*, and the *Pattern* interior style is supported by the server. The list of supported interior styles is determined by getting the *InteriorStyle* enumerated type values with request **PEXGetEnumeratedTypeInfo**. When interior style *Pattern* is supported and selected, the values of the pipeline state's *interior style index* and *bf interior style index* select the desired entries of the pattern table.

If the server does not support the *Pattern* interior style, it is not required to support the pattern table for use with a renderer resource. If the pattern table is not supported, **PEXCreateLookupTable**, **PEXGetTableInfo** and **PEXGetPredefinedEntries** will generate a *LookupTable* error.

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535.

Valid pattern table indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: PATTERN_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | COLOR_TYPE | color type |
| 2 | CARD16 | number of columns (*m*) |
| 2 | CARD16 | number of rows (*n*) |
| 2 | | unused |
| *m*n*sc*†2 | LISTofCOLOR | cell colors (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.
†2) The 2D array is flat with the first row of colors followed by the second row of colors and so on.

### 7.2.15 Text Bundle Table 5.0

Text bundle lookup tables contain collections of text attribute values.

**Encoding: Text Bundle Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *text_font_index* 5.0 | | | |
| | 4 | TABLE_INDEX | text font index (§6.9.93) | 1 |
| 0x00000002 | *text_precision* 5.0 | | | |
| | 4 | TEXT_PRECISION | text precision (§6.9.95) | *String* |
| 0x00000004 | *character_expansion* 5.0 | | | |
| | *fp* | FLOAT | char expansion factor (§6.9.20) | 1.0 |
| 0x00000008 | *character_spacing* 5.0 | | | |
| | *fp* | FLOAT | character spacing (§6.9.22) | 0.0 |
| 0x00000010 | *text_color* 5.0 | | | |
| | 4+*sc* | COLOR_SPECIFIER | text color (see COLOR for *sc*) (§6.9.91) | (*Indexed*, 1) |

†1) TEXT_TABLE_BITMASK : BITMASK

The text bundle table is accessed during rendering whenever the aspect source flag of one of the text attributes stored in the text bundle table is *Bundled* (§2.1.4.1). The aspect source flag for each attribute is a member of the pipeline state. The value of the pipeline state's *text bundle index* selects the desired entry of the table. The value of the bundled text attributes in that entry are then used to set the values of those text attributes.

For more information about each text attribute, see the corresponding text attribute output command (the section numbers of the text bundle table values are given with the field name in the encoding above). (These output commands set pipeline state which is used whenever the aspect source flag for a text attribute is *Individual* rather than *Bundled.*)

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an unsupported text precision is given or if a text font index of zero is given.

Valid text bundle indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: TEXT_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | TABLE_INDEX | text font index |
| 2 | TEXT_PRECISION | text precision |
| *fp* | FLOAT | character expansion factor |
| *fp* | FLOAT | character spacing |
| 4+*sc* | COLOR_SPECIFIER | text color (see COLOR for *sc*) |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.16 Text Font Table

5.0

Each entry of a text font table defines a font group.

**Encoding: Text Font Table Values**

| Bitmask†1 | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *font_group* 5.0 | | | |
| | 4 | CARD32 | number of font identifiers (*n*) | (imp-dep)†2 |
| | 4**n* | LISTofFONT_ID | font identifiers | (imp-dep)†2 |

†1) TEXT_FONT_TABLE_BITMASK : BITMASK
†2) The fallback must select some font capable of representing character set ISO-8859/1 (Latin1).

A font group is a numbered list, starting at one, of resource IDs of fonts. Each font resource ID is created by request **PEXOpenFont** (§ 13.2.4), or, if a PEX font is also supported by the X server, a font resource ID created by X11 request **OpenFont** is also acceptable. All fonts in the group must be in the list of fonts supported by the PEX server returned by requests **PEXListFonts** (§ 13.2.2) and **PEXListFontsWithInfo** (§ 13.2.3). Indicate a missing font in the group by using **None**, the *Null* resource identifier value defined by X.

The text font bundle table is accessed whenever a text or annotation text primitive is rendered. The value of the pipeline state's *text font index* selects the desired entry of the table. The *char_set* field of each mono-encoding of the primitive then selects the desired member of the font group.

If a font resource referred to in a font table is closed, the value *AlreadyFreed* (§ 3.7.7.120) is returned for the font ID in response to **PEXGetTableValues** and **PEXGetTableEntries**.

A *Value* error is issued if any member of the font group is other than a PEX font resource ID, the ID of an X11 font resource supported by PEX, or **None**.

Valid text font table indices are in the range [1, 65535]. The fallback entry is entry 1.

**Encoding: TEXT_FONT_TABLE_ENTRY†1**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of font identifiers (*n*) |
| 4**n* | LISTofFONT_ID | font identifiers |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

### 7.2.17 Texture Mapping Table

Each entry of a texture mapping table holds binds a set of attributes to a referenced texture map resource. The same texture map resource may be referenced by multiple texture mapping table entries.

**Encoding: Texture Mapping Table Values**

| Bitmask[†1] | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *texture_map_ids* | | | |
| | 2 | CARD16 | number of resources (*n*) | 0 |
| | 2 | | unused | |
| | 4**n* | LISTofTEXTURE_MAP_ID | list of texture map resources | |
| 0x00000002 | *rendering_control* | | | |
| | 4 | ENUM_TYPE_INDEX | rendering control | *PostLight* |
| 0x00000004 | *parameterization_method* | | | |
| | 4 | ENUM_TYPE_INDEX | parameterization method | *ReflectionSphereVRC* |
| 0x00000008 | *fp_data_index*[†2] | | | |
| | 4 | CARD16 | floating-point data index | 0 |
| 0x00000010 | *reflection_matrix*[†3] | | | |
| | 16**fp* | MATRIX | reflection matrix | *Identity* |
| | 4 | MATRIX_HINTS | reflection matrix hints | *identity* |
| 0x00000020 | *orientation_matrix* | | | |
| | 16**fp* | MATRIX | orientation matrix | *Identity* |
| | 4 | MATRIX_HINTS | orientation matrix hints | *identity* |
| 0x00000040 | *composition_method* | | | |
| | 4 | ENUM_TYPE_INDEX | composition method | *Replace* |
| 0x00000080 | *channel number*[†4] | | | |
| | 4 | CARD16 | channel selector | *TMChannelRed* |
| 0x00000100 | *environment_color*[†5] | | | |
| | 4+*sc* | COLOR_SPECIFIER | environment color | {*RGBFloat*, (0.0, 0.0, 0.0)} |
| 0x00000200 | *background_color*[†6] | | | |
| | 4+*sc* | COLOR_SPECIFIER | background color | {*RGBFloat*, (0.0, 0.0, 0.0)} |
| 0x00000400 | *sample_min_and_mag* | | | |
| | 2 | ENUM_TYPE_INDEX | minification sample method | *SingleBase* |
| | 2 | ENUM_TYPE_INDEX | magnification sample method | *SingleBase* |
| 0x00000800 | *boundary_conditions* | | | |
| | 2 | ENUM_TYPE_INDEX | t0 boundary condition | *Wrap* |
| | 2 | ENUM_TYPE_INDEX | t1 boundary condition | *Wrap* |
| | 2 | ENUM_TYPE_INDEX | t2 boundary condition | *Wrap* |
| | 2 | | unused | |
| 0x00001000 | *clamping_method*[†7] | | | |
| | 4 | ENUM_TYPE_INDEX | boundary clamp method | *Discontinue* |
| 0x00002000 | *clamp_color*[†8] | | | |
| | 4+*sc* | COLOR_SPECIFIER | boundary clamp color | {*RGBFloat*, (0.0, 0.0, 0.0)} |

**Encoding: Texture Mapping Table Values**

| Bitmask[†1] | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| **0x00004000** | ***sampling_hints*** 5.2 | | | |
| | *fp* | FLOAT | depth sampling bias hint | 0.0 |
| | *fp* | FLOAT | t0 frequency hint | 1.0 |
| | *fp* | FLOAT | t1 frequency hint | 1.0 |
| | *fp* | FLOAT | t2 frequency hint | 1.0 |

†1) TEXTURE_MAP_TABLE_BITMASK : BITMASK
†2) Field *fp data index* is ignored unless field *parameterization method* selects method *FloatData*.
†3) Field *reflection matrix* is ignored unless field *parameterization method* selects method *ReflectionSphereVRC* or *ReflectionSphereWC*.
†4) Field *channel number* is ignored unless field *composition method* selects method *BlendEnvColor*.
†5) Field *environment color* is ignored unless field *composition method* selects method *BlendEnvColor*.
†6) Field *background color* is ignored unless field *composition method* selects method *DecalBackground*.
†7) Field *clamping method* is ignored unless field *boundary conditions* selects condition *ClampExplicitMethod*.
†8) Field *clamp color* is ignored unless field *boundary conditions* selects condition *ClampExplicitMethod* and field *clamping method* selects method *Color*.

### 7.2.17.1 Selecting and Applying a Texture

The texture mapping table is accessed when rendering surface primitives if the pipeline state's *interior style* is *Texture* and the pipeline state's *active textures* field is non-empty. If the pipeline state's *facet distinguish flag* is *True*, the texture mapping table is accessed when rendering back-facing facets if the pipeline state's *bf interior style* is *Texture* and the pipeline state's *bf active textures* field is non-empty. The active textures lists select entries of the texture mapping table which each represent a texture map and an associated set of attributes. If the number of entries in the active textures list exceeds the maximum number of texture maps which can be applied to a single facet, the texture maps beyond the first *MaxTextureMaps* indices will be ignored. Use request **PEXGetImpDepConstants (§4.1.4)** to inquire *MaxTextureMaps*. If the active textures list is empty, the default table values are used to apply the single default texture map.

Not all the texture mapping table fields are applied to every texture map. The descriptions of the fields below indicate if, and how, the fields are applied to the texture map.

Texture mapping table field *texture map ids* is used to reference the texture map resources which define the texture map. Texture map resources are discussed in Chapter 14. The number of texture map resources needed to define a single texture map is determined by the texture mapping table's *parameterization method* field. All currently defined parameterization methods use a single texture map resource to define a texture map.

There is a default texture map used if the *texture map ids* fallback (an empty list) is used. The default texture map appears as a black and white checkerboard of 8x8 squares. The checkerboard is defined by a 2D array of 16x16 texels, with each square defined by 2x2 texels. The square at t0=t1=0.0 is white.

The texture mapping table's *rendering control* field controls where in the rendering pipeline the texture map is applied. The currently defined values, shown in Table 7-8 7, interact heavily with the surface interpolation method (§6.9.86) and texture mapping sample frequency (§6.9.98). Rendering pipeline diagrams are provided in Chapter 2 to help clarify these interactions.

**Table 7-8 . Enumerated Type *TMRenderingControl***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *PreLight* | The texture map is to be applied to the primitive before the reflectance calculations. |
| 1 | *PostLight* | The texture map is to be applied to the primitive after the reflectance calculations. |

### 7.2.17.2 Parameterization Controls

The parameterization attributes control how the texture map coordinates are derived. There are texture map coordinates associated with each point on the surface for which a texture map value is sampled. They are used to select the point within the texture map which is sampled. There is one texture map coordinate for each dimension of the texture map. For example, three texture map coordinates, represented as (t0, t1, t2), are used to sample a 3D texture map. The texture coordinates are normalized, that is, they range from 0.0 to 1.0 across the texture map.

#### 7.2.17.2.1 Parameterization Method

The *parameterization method* field of the texture mapping table defines how texture map coordinates are derived: whether explicitly defined by the primitive's data, calculated through a projection mapping, or determined through some other means. The defined parameterization methods are listed in Table 7-9 .

**Table 7-9 . Enumerated Type *TMParameterizationMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *VertexMCCoord*†1 | The texture map coordinates are set equal to the MC space values of the vertex coordinates: t0 = *x*; t1 = *y*; t2 = *z*. |
| 1 | *VertexWCCoord*†1 | The texture map coordinate values are set equal to the WC space values of the vertex coordinates: t0 = *x*'; t1 = *y*'; t2 = *z*'. |
| 2 | *VertexMCNormal*†1 | The texture map coordinates are set equal to the MC space value of the vertex normal. If a normal is not explicitly supplied with the vertex, the calculated geometric normal is used instead. t0 = *nx*; t1 = *ny*; t2 = *nz*. |
| 3 | *VertexWCNormal*†1 | The texture map coordinates are set equal to the WC space value of the vertex normal. If a normal is not explicitly supplied with the vertex, the calculated geometric normal is used instead. t0 = *nx*'; t1 = *ny*'; t2 = *nz*'. |
| 4 | *NURBUV* | The texture map coordinates are set equal to the *u* and *v* parameters of the non-uniform b-spline surface: t0 = u; t1 = v; t2 = 0.0.<br><br>If the primitive is not a non-uniform b-spline surface, the texture map coordinates are set to: t0 = t1 = t2 = 0.0. |
| 5 | *FloatData* | The texture map coordinates are members of the vertex's floating-point data list. The *fp data index* field of the texture mapping table is the index into the floating-point data list of the value which is assigned to t0. The subsequent values in the floating-point data list are assigned to t1 and t2:<br><br>t0 = vertex_fp_data[*fp data index*]<br>t1 = vertex_fp_data[*fp data index* + 1] (if 2D map)<br>t2 = vertex_fp_data[*fp data index* + 2] (if 3D map)<br><br>If the primitive is a non-uniform b-spline surfaces, the values are fetched from the spline's data splines:<br>t0 = first value of the data spline selected by the *fp data index*<br>t1 = first value of the data spline selected by *fp data index* + 1<br>t2 = first value of the data spline selected by *fp data index* + 2<br><br>If the number of floating-point data values is insufficient for the dimension of the texture map, the texture coordinate values used are implementation-dependent. |
| 6 | *ReflectionSphereVRC* | See §7.2.17.2.2. |
| 7 | *ReflectionSphereWC* | See §7.2.17.2.3. |

†1) In the case of rendering primitives which do not include explicit vertices, but may have computed vertices, the behavior is implementation-dependent. The implementation may use the computed vertices or may evaluate the coordinate or normal at each pixel.

The currently defined projection mapping parameterization methods, *ReflectionSphereVRC* and *ReflectionSphereWC*, project a reflection vector (the 3D "source texture map coordinates") onto a sphere (the "projection object") to determine the 3D texture map coordinates (t0,t1,t2=0.0).

The reflection sphere has a fixed orientation, but the reflection vector is transformed by the *reflection matrix*, another field of the texture mapping table. The reflection transformation is accomplished prior to projection, thus effectively orienting the projection in the space in which it is defined. The *reflection matrix* is expected to describe only rotations and/or reflections, or to have no effect by virtue of being an identity matrix. The results re implementation-dependent if the *reflection matrix hints* contain other than *identity*, *rotation*, or *reflection*, or if the matrix does not conform to its hints. See §6.9.34 for the definition of the matrix hints values.

#### 7.2.17.2.2 Spherical Reflection in VRC Projection Mapping

The infinite sphere projection object has (0,0,0) as its origin and the +*Y*-axis as its axis of revolution. The texture seams lie on the positive and negative *X* axes. The source texture map coordinates are a reflection vector (s0, s1, s2), computed to determine a point on the interior of the sphere.

The reflection vector is calculated as follows:

$$s = i - 2n\,(n \text{ dot } i)$$ **Equation 7.6**

where *s* is the reflection vector, *n* is the vertex normal (explicit or calculated), and *i* is the unit vector from the eyepoint (computed as described in §7.2.18) to the vertex. This calculation is performed in VRC space, after the vertex and its normal have been transformed to VRCs. After this, the texture coordinates are derived as follows:

$$t_0 = \frac{s_0}{m} + \frac{1}{2}$$

$$t_1 = \frac{s_1}{m} + \frac{1}{2}$$

$$\text{where: } m = 2 \times \sqrt{s_0^2 + s_1^2 + (s_2 + 1)^2}$$ **Equation 7.7**

#### 7.2.17.2.3 Spherical Reflection in WC Projection Mapping

The source texture map coordinates, a reflection vector (s0,s1,s2), are normalized and conceptually projected onto the infinite sphere (with (0,0,0) as its origin) projection object. The axis of revolution of the sphere is the +*Y* axis (in WC space). The texture seam sweeps from the +*X* axis (in WC space) in a counterclockwise direction: [0..2PI].

The reflection vector is calculated as follows:

$$s = -(i - 2n\,(n \cdot i))$$ **Equation 7.8**

where *s* is the reflection vector, *n* is the vertex normal (explicit or calculated), and *i* is the unit vector from the eyepoint (computed as described in §7.2.18) to the vertex. This calculation is performed in WC space, after the vertex and its normal have been transformed to WC space. Note this parameterization method is primarily intended for use with a perspective, as opposed to orthogonal, view projection. After the reflection vector is calculated, the texture coordinates are derived as follows.

$$
\begin{aligned}
t_0 &= 0.75 \quad : s_0 = 0.0, s_2 > 0.0 \\
&= 0.25 \quad : s_0 = 0.0, s_2 \le 0.0 \\
&= \frac{-atan\left(\frac{s_2}{s_0}\right)}{2\pi} + 0.5 \quad : s_0 < 0.0 \\
&= \frac{-atan\left(\frac{s_2}{s_0}\right)}{2\pi} + 1.0 \quad : s_0 > 0.0, s_2 > 0.0 \\
&= \frac{-atan\left(\frac{s_2}{s_0}\right)}{2\pi} + 0.0 \quad : s_0 > 0.0, s_2 \le 0.0 \\
t_1 &= \frac{asin\,(s_1)}{\pi} + 0.5 \\
&\text{where:}\left(\frac{-\pi}{2} \le atan\,(r) \le \frac{\pi}{2}\right)\text{and}\left(\frac{-\pi}{2} \le asin\,(r) \le \frac{\pi}{2}\right)
\end{aligned}
$$

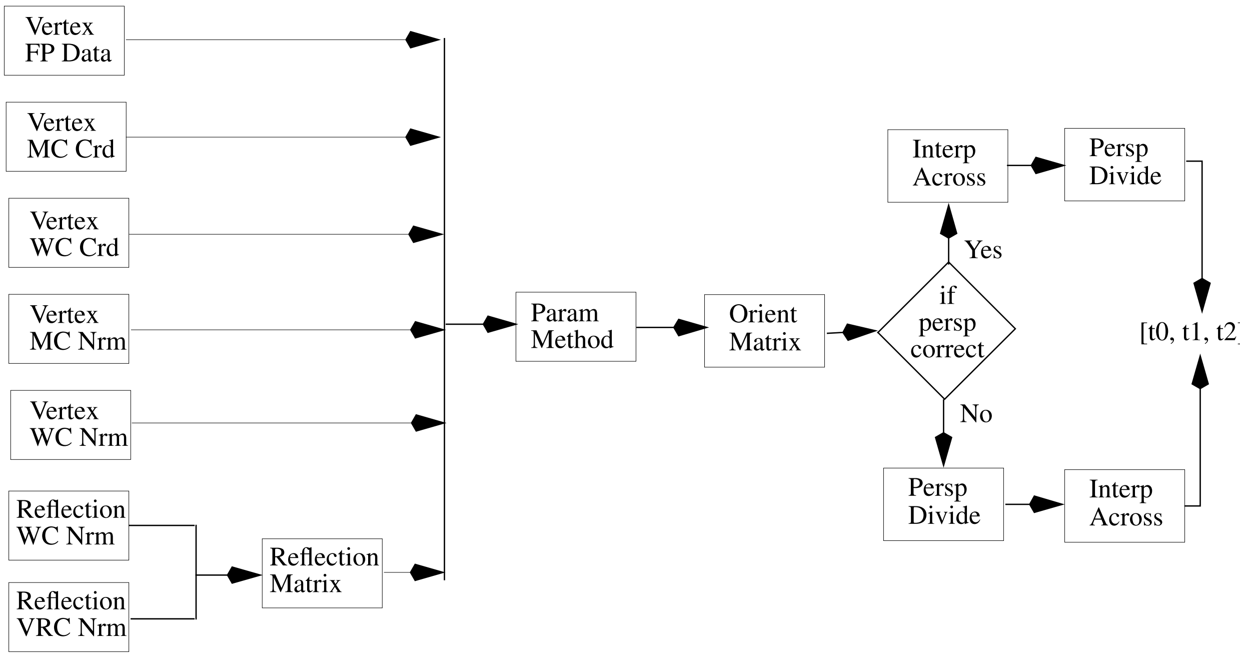
**Equation 7.9**

### 7.2.17.2.4 Orientation Transformation, Interpolation, and Perspective Correction

Once the texture map coordinates are determined, independent of which parameterization method was used, they are transformed by the *orientation matrix* field of the texture mapping table. The orientation matrix hints provide opportunity to optimize the transformation. See (§6.9.34) for the usage and definition of matrix hints. The orientation transformation occurs before any interpolation of the texture map coordinates across the facet that may be required before the coordinates are used to sample the texture map. If the projection is perspective, the interpolated texture map coordinates are also adjusted according to the pipeline state's *tm perspective correction* (§6.9.96) before the sampling is done.

§Figure 7-1 provides a graphical description of parameterization.

**Figure 7-1 . Texture Parameterization**




### 7.2.17.3 Composition Controls

The currently defined composition attributes control how the texture map color interacts with the primitive's color to determine the primitive's color after the texture map is applied.

The texture mapping table *composition method* field specifies how the texture map is blended with the primitive's existing data. The currently defined composition methods (Table 7-10 ) assume two things:

- The texture is a color domain texture, i.e. it modifies intrinsic color (including alpha) of rendered primitives.
- The rendering color model is *RGB*. The current methods have undefined behavior for rendering color models other than *RGB*.

**Table 7-10 . Enumerated Type *TMCompositionMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *Replace* | §7.2.17.3.1 . |
| 1 | *Modulate* | §7.2.17.3.2 . |
| 2 | *BlendEnvColor* | §7.2.17.3.3 . |
| 3 | *Decal* | §7.2.17.3.4 . |
| 4 | *DecalBackground* | §7.2.17.3.5 . |

For each composition method below, equations are given. The equations given are for the input into each texture map *i* and show the $C_{out}$ and $A_{out}$ for each texture map *i*. The output of each texture map i becomes the input of the next texture map *i+1*. The final output of the texture mapping stage is the output of texture map *n*, where there are *n* texture maps active for the stage. Note in some *PreLight* rendering control cases there are additional values used as input into each texture map *i* (e.g., $C_W$) and used as output from the texture mapping stage (e.g., $C_S$). See the rendering pipeline diagrams in Chapter 2 for clarification. Table 7-10 defines the symbols used to define the composition methods.

**Table 7-11 . Symbols Used in Defining the Composition Methods**

| Symbol | Description |
|---|---|
| *in* | Primitive color components input into texture *i* of a texture mapping stage. |
| *out* | Primitive color components output from texture *i* of a texture mapping stage. |
| $R$ | The red component. |
| $G$ | The green component. |
| $B$ | The blue component. |
| $C$ | *R, G,* and *B* considered together. |
| $A$ | The alpha component. |
| $L$ | The luminance component. |
| $C_W$ | A portion of primitive color input into the *PreLight* texture mapping stage. $C_W$ is derived in the preceding lighting stage by conducting the ambient and diffuse terms of the reflectance equation assuming the primitive's color is white.<br>The same value of $C_W$ is input into each texture *i* of the texture mapping stage; it is not modified by texture mapping. |
| $C_D$ | Another portion of primitive color input into the *PreLight* texture mapping stage. $C_D$ is the ambient and diffuse components of color derived in the preceding lighting stage by conducting the reflectance equation using the primitive's color (as opposed to assuming the primitive is white). |
| $C_S$ | A third portion of primitive color input into the *PreLight* texture mapping stage. $C_S$ is the specular component of color derived in the preceding lighting stage by conducting the reflectance equation using the primitive's color (as opposed to assuming the primitive is white).<br>$C_S$ is not affected by the texture mapping stage. It is combined with the $C_D$ output from the texture mapping stage to form the RGB components of the primitive color output from the texture mapping stage. |
| $C_T$ | *R, G,* and *B* of the texture map color considered together. If the texels have a luminance component instead of *RGB* components, $R_T = G_T = B_T = L1$ |
| $A_T$ | The alpha component of the texture map color. |
| $C_E$ | When the composition method is *BlendEnvColor*, $C_E$ is taken from the *environment color* field of the texture mapping lookup table. |
| $C_B$ | When the composition method is *DecalBackground*, $C_B$ is taken from the *background color* field of the texture mapping lookup table. |
| $\Gamma$ | When the composition method is *BlendEnvColor*, $\Gamma$ is taken from the texture map color. For texels defined with *RGB* or *RGBA* components: if the texture mapping lookup table's *channel number* is *TMChannelRed*, $\Gamma = R_T$; if *channel number* is *TMChannelGreen*, $\Gamma = G_T$; if *channel number* is *TMChannelBlue*, $\Gamma = B_T$, and if there is an alpha component and *channel number* is *TMChannelAlpha*, $\Gamma = A_T$. For texels defined with a luminance component: $\Gamma = L$<br>The channel number constants are defined in Table 7-10 . |

**Table 7-12 . Channel Selection Constant Values†1**

| Value | Symbol | Description |
|---|---|---|
| 0 | *TMChannelRed* | Select the red channel, channel 0. |
| 1 | *TMChannelGreen* | Select the green channel, channel 1. |
| 2 | *TMChannelBlue* | Select the blue channel, channel 2. |
| 3 | *TMChannelAlpha* | Select the alpha channel, channel 3. |

†1) Where this is overlap, the entries of this table match the entries of Table 14-4 (§14.1.2), which defines the channel selectors for request **PEXCreateColorMipmapTMFromResources**.

### 7.2.17.3.1 Composition Method Replace

The texture map data unconditionally replaces the primitive’s color.

**Table 7-13 . Composition Method *Replace***

<table>
<tr><th>Texel Components</th><th>Rendering Control</th><th>Interpolation Method</th><th>Sample Frequency</th><th>$C_{out}$</th><th>$A_{out}$</th></tr>
<tr><td rowspan="4">RGB, L</td><td rowspan="3">PreLight</td><td rowspan="2">None, Color</td><td>Pixel</td><td>$C_W \bullet C_T$ and $C_S$</td><td>1.0</td></tr>
<tr><td>InterpDep</td><td rowspan="3">$C_T$</td><td rowspan="3">1.0</td></tr>
<tr><td>DotProduct, Normal</td><td>(any)</td></tr>
<tr><td>PostLight</td><td>(any)</td><td>(any)</td></tr>
<tr><td rowspan="4">RGBA, LA</td><td rowspan="3">PreLight</td><td rowspan="2">None, Color</td><td>Pixel</td><td>$C_W \bullet C_T$ and $C_S$</td><td>$A_T$</td></tr>
<tr><td>InterpDep</td><td rowspan="3">$C_T$</td><td rowspan="3">$A_T$</td></tr>
<tr><td>DotProduct, Normal</td><td>(any)</td></tr>
<tr><td>PostLight</td><td>(any)</td><td>(any)</td></tr>
</table>

### 7.2.17.3.2 Composition Method Modulate

The texture map color and primitive color are blended.

**Table 7-14 . Composition Method *Modulate***

| Texel Components | Rendering Control | Interpolation Method | Sample Frequency | $C_{out}$ | $A_{out}$ |
|---|---|---|---|---|---|
| RGB, L | *PreLight* | *None, Color* | *Pixel* | $C_D \bullet C_T$ and $C_S$ | $A_{in}$ |
| | | | *InterpDep* | $C_{in} \bullet C_T$ | $A_{in}$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |
| RGBA, LA | *PreLight* | *None, Color* | *Pixel* | $C_D \bullet C_T$ and $C_S$ | $A_{in} \bullet A_T$ |
| | | | *InterpDep* | $C_{in} \bullet C_T$ | $A_{in} \bullet A_T$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |

### 7.2.17.3.3 Composition Method Blend Environment Color

The primitive color and a constant environment color are blended by the texture map color. The primitive alpha is blended with the texture map alpha, if the texture map has alpha.

**Table 7-15 . Composition Method *BlendEnvColor***

| Texel Components | Rendering Control | Interpolation Method | Sample Frequency | $C_{out}$ | $A_{out}$ |
|---|---|---|---|---|---|
| RGB, L | *PreLight* | *None, Color* | *Pixel* | $C_D(1-\Gamma) + C_E\Gamma$ and $C_S$ | $A_{in}$ |
| | | | *InterpDep* | $C_{in}(1-\Gamma) + C_E\Gamma$ | $A_{in}$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |
| RGBA, LA | *PreLight* | *None, Color* | *Pixel* | $C_D(1-\Gamma) + C_E\Gamma$ and $C_S$ | $A_{in} \bullet A_T$ |
| | | | *InterpDep* | $C_{in}(1-\Gamma) + C_E\Gamma$ | $A_{in} \bullet A_T$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |

#### 7.2.17.3.4 Composition Method Decal

The texture map color and primitive color are blended by the texture map alpha. If the texture map doesn’t contain alpha, the texture map color replaces the primitive color. In either case, the primitive’s alpha is unchanged.

**Table 7-16 . Composition Method *Decal***

| Texel Components | Rendering Control | Interpolation Method | Sample Frequency | $C_{out}$ | $A_{out}$ |
|---|---|---|---|---|---|
| RGB, L | *PreLight* | *None, Color* | *Pixel* | $C_W \bullet C_T$ and $C_S$ | $A_{in}$ |
| | | | *InterpDep* | $C_T$ | $A_{in}$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |
| RGBA, LA | *PreLight* | *None, Color* | *Pixel* | $C_D(1-A_T) + C_W \bullet C_T \bullet A_T$ and $C_S$ | $A_{in}$ |
| | | | *InterpDep* | $C_{in}(1-A_T) + C_T A_T$ | $A_{in}$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |

#### 7.2.17.3.5 Composition Method DecalBackground

The texture map color and background color are blended according to the texture map alpha, and this blend replaces the primitive color. If the texture map doesn’t contain alpha, the texture map color replaces the primitive color. In either case, the primitive’s alpha is unchanged.

**Table 7-17 . Composition Method *DecalBackground***

| Texel Components | Rendering Control | Interpolation Method | Sample Frequency | $C_{out}$ | $A_{out}$ |
|---|---|---|---|---|---|
| RGB, L | *PreLight* | *None, Color* | *Pixel* | $C_W \bullet C_T$ and $C_S$ | $A_{in}$ |
| | | | *InterpDep* | $C_T$ | $A_{in}$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |
| RGBA, LA | *PreLight* | *None, Color* | *Pixel* | $C_B(1-A_T) + C_W \bullet C_T \bullet A_T$ and $C_S$ | $A_{in}$ |
| | | | *InterpDep* | $C_B(1-A_T) + C_T A_T$ | $A_{in}$ |
| | | *DotProduct, Normal* | (any) | | |
| | *PostLight* | (any) | (any) | | |

### 7.2.17.4 Sampling Controls

The sampling attributes are used to determine how a texture map color is derived when the texture map is sampled.

#### 7.2.17.4.1 Minification and Magnification Sample Methods

The *sample min and mag* field of the texture mapping table defines the minification and magnification sample methods. These methods specify how texels are sampled from the texture map and mapped to primitive pixels when the area covered by a texel and pixel differ. The sample methods are defined in Table 7-18 .

In the case of a pixel overlapping more than one texture map texel, the texture minification method indicates how the sample value is derived for that pixel.

In the case of a pixel overlapping less than one texture map texel, the texture magnification method indicates how the sample value is derived for that pixel.

**Table 7-18 . Enumerated Type *TMSampleMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *SingleBase* | The closest single texel selected by the texture coordinate(s) is sampled from the base texture map level. |
| 1 | *WeightedBase* | The $2^n$ (where n is the dimension of the texture map) closest texels[†1] selected by the texture coordinate(s) are sampled from the base texture map level. Their weighted average is used. |
| 2 | *SingleInMipmap* | The closest single texel selected by the texture coordinate(s) is sampled from the closest texture map level to the sample depth. |
| 3 | *WeightedInMipmap* | The $2^n$ (where n is the dimension of the texture map) closest texels[†1] selected by the texture coordinate(s) are sampled from the closest texture map level to the sample depth. Their weighted average is used. |
| 4 | *SingleBetweenMipmaps* | The closest texel selected by the texture coordinate(s) is sampled from the two closest texture map levels to the sample depth. The texel found at the exact sample depth by interpolation between these two sampled texels is used. If the sample depth is beyond the base or pinnacle texture map levels, that level is used and this method behaves the same as *SingleInMipmap*. |
| 5 | *WeightedBetweenMipmaps* | The $2^n$ (where n is the dimension of the texture map) closest texels[†1] selected by the texture coordinate(s) are sampled from the two closest texture map levels to the sample depth. A weighted average is taken of these $2^n$ texels on each of the texture map levels. The texel found at the exact sample depth by interpolation between these two calculated texels is used. If the sample depth is beyond the base or pinnacle texture map levels, that level is used and this method behaves the same as *WeightedInMipmap*. |

†1) The selected texels may include texels synthesized from boundary conditions (§7.2.17.4.2). A boundary condition of *ClampExplicitMethod* with clamp method *Discontinue* does not contribute to the weighted average.

#### 7.2.17.4.2 Boundary Conditions

The *boundary conditions* field of the texture mapping table specifies the texturing to be applied to the primitive when the texture coordinates (t0, t1, t2) select a point beyond a boundary of the texture map. That is, the calculated texture coordinates are outside the range [0.0,1.0]. The registered boundary conditions are listed in Table 7-19 . The boundary condition in each direction is independently controlled.

**Table 7-19 . Enumerated Type *TMBoundaryCondition***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *ClampExplicitMethod* | The method specified by the *clamping method* field of the texture mapping table is applied. |
| 1 | *ClampBorder* | The closest boundary texel is used. |
| 2 | *Wrap* | Texel sampling wraps back to the opposite texture map border, creating a “tiling” or “repeating” effect of the texture map across the primitive. |
| 3 | *Mirror* | Texel sampling is reversed across the texture map, creating a “mirrored alternating” pattern of the texture map across the primitive. |

### 7.2.17.4.3 Clamping Method

The *clamping method* field of the texture mapping table specifies whether to discontinue texturing or to use an explicit color when *boundary condition ClampExplicitMethod* is selected. Although these are seemingly really additional boundary conditions, they are controlled through a separate field of the texture mapping table by design. With this level of direction, it is always clear what the behavior should be when a location is beyond the texture map in more than one direction, even if the boundary conditions for those directions differ.

**Table 7-20 . Enumerated Type *TMBoundaryClampMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *Discontinue* | Texturing is discontinued. That is, the primitive’s appearance before applying the current texture map is maintained. |
| 1 | *Color* | The color specified by the *clamp color* field of the texture mapping table is used. |

### 7.2.17.4.4 Sampling Hints

The texture mapping table *sampling hints* field is used to adjust the selected sampling location. The depth bias hint is used to adjust the sampling depth. A bias of -1.0 moves the sampling depth one level toward the texture map’s base level, effectively making the texturing more detailed, or jaggy, than it would otherwise have been. A bias of 2.0, for example, moves the sampling depth away from the texture map’s base level by two levels, effectively blurring the texturing. The following is suggested formulae for implementations; implementations may use other formulae, but should be careful to achieve the same effect as the suggested formulae.

$$\begin{aligned} A &= max\left( t_0 freq \times \frac{dt_0}{dx}, t_1 freq \times \frac{dt_1}{dx} \right) \\ B &= max\left( t_0 freq \times \frac{dt_0}{dy}, t_1 freq \times \frac{dt_1}{dy} \right) \\ depth &= bias + 0.5 \times log2\left( A^2 + B^2 \right) \end{aligned}$$

**Equation 7.10**

The extension of the suggested formulae to three-dimensional texture maps is left unspecified.

The suggested depth bias formulae utilize the frequency hints. There is a frequency hint corresponding to each direction of the texture map. A frequency hint is set to something less than 1.0, but greater than 0.0, if blurring in that direction would be unacceptably high due to low spatial frequency A value of 1.0 for the corresponding texture map coordinate's frequency hint indicates the spatial frequency isn't high. If the spatial frequency of the texture map data is unknown, it is best to set the values to 1.0.

Inquire *TMSamplingHintsUsed* with request **PEXGetImpDepConstants** (§4.1.4) to determine if the sampling hints are used.

### 7.2.17.5 Restrictions and Errors

The superseded requests **PEXSetTableEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** do not operate on texture mapping tables (§7.4).

Out of range direct color components are treated in an implementation-dependent way. A *ColorType* error is issued if an unsupported color type is given. A *Value* error is issued if the color type is *Indexed* and the color index is 65535. A *Value* error is also issued if an unsupported value is given for any of rendering control, parameterization method, reflection matrix hints, orientation matrix hints, channel selector, composition method, minification or magnification sample method, t0, t1 or t2 boundary condition, or boundary clamp method.

Valid texture mapping table indices are in the range [1, 65535]. The fallback entry is entry 1.

### 7.2.18 View Table 5.0

Each entry of a view table holds a view definition.

**Encoding: View Table Values**

| Bitmask[†1] | # of Bytes | Data Type or Value | Description | Fallback |
|---|---|---|---|---|
| 0x00000001 | *clip_flags* 5.0 | | | |
| | 4 | | clip flags | all flags *On* |
| | | 0x0001 *clip_xy* | | |
| | | 0x0002 *clip_back* | | |
| | | 0x0004 *clip_front* | | |
| 0x00000002 | *clip_limits* 5.0 | | | |
| | 6**fp* | NPC_SUBVOLUME | clip limits | (0,0,0)-(1,1,1) |
| 0x00000004 | *orientation_matrix* 5.0 | | | |
| | 16**fp* | MATRIX | view orientation matrix | *Identity* |
| | 4 | MATRIX_HINTS | view orientation matrix hints[†2] | *Identity* |
| 0x00000008 | *mapping_matrix* 5.0 | | | |
| | 16**fp* | MATRIX | view mapping matrix | *Identity* |
| | 4 | MATRIX_HINTS | view mapping matrix hints[†2] | *Identity* |
| 0x00000010 | *eyepoint*[†3] 5.2 | | | |
| | 4+3**fp* | EYEPOINT (see below) | eyepoint | (*Implicit*, n/a) |

†1) VIEW_TABLE_BITMASK : BITMASK

†2) The requests **PEXSetTableEntries** request sets the matrix hints to *None*, and the **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** requests do not return the matrix hints.

†3) The request **PEXSetTableEntries** sets the eyepoint to *Implicit*, and the **PEXGetTableEntries**, **PEXGetTableEntry**, and **PEXGetPredefinedEntries** requests do not return the eyepoint.

*clip_flags* contains three bits that indicate whether primitives should be clipped against the sides, back, and front planes of the volume specified by *clip_limits*. The *clip_limits* specify the minimum and maximum of a rectangular volume in NPCs. Together these define the view clipping volume. The actual clipping volume for each view is a rectangular parallelpiped that is computed by taking the intersection of the view clipping volume and the NPC subvolume (§3.7.7.87).

The *orientation_matrix* defines the mapping from WCs to VRCs. The *mapping_matrix* defines the mapping from VRCs to NPCs (§2.2.1). Together the *orientation_matrix* and *mapping_matrix* define the view transformation. The matrix hints provide opportunity to optimize this transformation. See §6.9.34 for the usage and definition of matrix hints.

*eyepoint* is a hint describing the definition (in WCs) of the eyepoint used with the view. *EyepointUsed,* inquired with request **PEXGetImpDepConstants**, indicates whether the server uses the eyepoint given. The eyepoint specification methods are shown in Table 7-21 .

**Table 7-21 . Eyepoint Specification Method Constant Values**

| Value | Symbol | Description |
|---|---|---|
| 1 | *Implicit* | The eyepoint is defined implicitly by the view. No additional data is defined with this method.<br>The eyepoint is obtained by transforming a point at infinite positive Z (NPCs), (0, 0, 1, 0), by the inverse of the view transformation. If the view matrix specifies perspective projection, a point is obtained. If parallel projection is in use, a direction vector is obtained. Singular view matrices lead to implementation-dependent results. |
| 2 | *Directional* | The eyepoint is located at infinity along a particular direction.<br>A vector (in WCs) anchored at the origin and pointing toward the eyepoint defines the eyepoint.<br>The *Directional* method should be used when the projection is parallel. It can also be used with faster but less accurate results when the projection is perspective. |
| 3 | *Positional* | The eyepoint is located at a particular position (in WCs).<br>The *Positional* method should be used only when the projection is perspective. |

The view table is always accessed during rendering. The value of the pipeline state's *view index* selects the desired entry of the table.

A *Value* error is issued if an unsupported eyepoint specification method is given, if an undefined bit is set in the *clip flags* field, or if a directional eyepoint has a direction vector of length zero.

Valid view table indices are in the range [0, 65534]. The fallback entry is entry 0.

**Encoding: VIEW_TABLE_ENTRY**†1

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | | | clip flags |
| | 0x0001 | *clip_xy* | |
| | 0x0002 | *clip_back* | |
| | 0x0004 | *clip_front* | |
| 2 | | | unused |
| 6**fp* | NPC_SUBVOLUME | | clip limits |
| 16**fp* | MATRIX | | orientation matrix |
| 16**fp* | MATRIX | | mapping matrix |

†1) Beware only fields defined prior to PEX version 5.2 are included in this type.

All geometric data is specified in modeling coordinates unless otherwise specified.

| | | |
|---|---|---|
| EYEPOINT | : | [method: EYEPOINT_METHOD,<br>x, y, z : FLOAT] |
| EYEPOINT_METHOD | : | {*Implicit, Directional, Positional*} |

**Encoding: Type EYEPOINT**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 2 | CARD16 | | eyepoint specification method |
| | 1 | *Implicit* | |
| | 2 | *Directional* | |
| | 3 | *Positional* | |
| *2* | | | unused |
| *fp* | FLOAT | | x†1 |
| *fp* | FLOAT | | y |
| *fp* | FLOAT | | z |

†1) *x, y,* and *z* are ignored if the method is *Implicit.* They define a directional vector if the method is *Directional.* They define a position if the method is *Positional.*

## 7.3 Lookup Table Requests

Lookup tables are X11 resources and carry all of the responsibilities and access rights of X11 resources. The requests described in this section are those that operate on lookup table resources. Requests that operate on lookup tables but that have been superseded are described in §7.4.

### 7.3.1 Change Table Values

5.2

**Name**

**PEXChangeTableValues**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *lut_id*: | LOOKUP_TABLE_ID |
| *index*: | TABLE_INDEX |
| *fields*: | TABLE_BITMASK |
| *values*: | LISTofVALUE |

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

Value
1) Index 0 (or 65535) is not valid for this table type.
2) One of the specified fields is illegal.
3) One or more of the values in *values* is not valid. (See the specification of the table types.)

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

ColorType
1) The specified color type is not supported.
2) The color type of a color table entry may not be *Indexed*.

Alloc
Defining this entry would exceed the table’s maximum number of entries.

**Description**

This request defines or sets the entry indicated by *index* in the lookup table identified by *lut_id*. If the entry is already defined, then each field indicated by *fields* is set to the corresponding value in *values*. Members that are not named in *fields* are not modified. If the entry is not defined, it is created and the requested fields are set as for a previously defined entry. Fields not named in *fields* are given their fallback value (§7.1.1). Floating-point values in *values* are in the floating-point format specified in *fp_format*.

**Encoding: PEXChangeTableValues**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 106 | PEX opcode |
| 2 | 4+*s*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | TABLE_INDEX | index |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 4 | TABLE_BITMASK | fields to change |
| *s*†1 | LISTofVALUE | values |

†1) s is the sum of the sizes of the *field* values. Each value is aligned on a 4-byte boundary.

### 7.3.2 Copy Lookup Table

**Name**

**PEXCopyLookupTable**

**Request**

| | |
|---|---|
| *src_lut_id*: | LOOKUP_TABLE_ID |
| *dest_lut_id*: | LOOKUP_TABLE_ID |

**Errors**

LookupTable
One of the lookup tables identifiers *src_lut_id* or *dest_lut_id* does not identify a valid lookup table.

Match
The lookup tables are either not of the same type or were not created for the same class of drawable.

**Description**

This request deletes all the entries in the lookup table identified by *dest_lut_id* and then copies to it all the entries in the lookup table identified by *src_lut_id*. The *dest_lut_id* must already exist as a valid resource and it must be of the same type as *src_lut_id*.

**Encoding: PEXCopyLookupTable**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 5 | PEX opcode |
| 2 | 3 | request length |
| 4 | LOOKUP_TABLE_ID | source lookup table ID |
| 4 | LOOKUP_TABLE_ID | destination lookup table ID |

### 7.3.3 Create Lookup Table

5.0

**Name**

**PEXCreateLookupTable**

**Request**

| | |
|---|---|
| *drawable_example*: | DRAWABLE_ID |
| *lut_id*: | LOOKUP_TABLE_ID |
| *table_type*: | TABLE_TYPE |

**Errors**

IDChoice
*lut_id* is already in use or not in the range assigned to the client.

Drawable
*drawable_example* does not identify a valid drawable.

Match
The drawable identified by *drawable_example* is not supported by the PEX implementation.

Value
*table_type* is not a valid table type.

Alloc
The server failed to allocate the table.

LookupTable
The requested table type is valid, but not supported by the PEX implementation.

**Description**

This request creates a lookup table resource for the specified *lut_id*, for use with drawables with the same root window, depth and visual ID as the example drawable specified by *drawable_example*. The *table_type* parameter indicates the type of lookup table to create.

**Encoding: PEXCreateLookupTable**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 4 | PEX opcode |
| 2 | 4 | request length |
| 4 | DRAWABLE_ID | example drawable ID |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 2 | TABLE_TYPE | table type |
| 2 | | unused |

### 7.3.4 Delete Table Entries

5.0

**Name**

**PEXDeleteTableEntries**

**Request**

| | |
|---|---|
| *lut_id*: | LOOKUP_TABLE_ID |
| *start*: | TABLE_INDEX |
| *count*: | CARD16 |

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

Value
1) *start* + *count* exceeds the maximum valid index for this table type.
2) Entry 0 (or 65535) is not valid for this table type.

**Description**

This request deletes entries in the lookup table identified by *lut_id*. The entries deleted are those whose index is between and including the entry indicated by *start* and the index *start*+*count*-1.

**Encoding: PEXDeleteTableEntries**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 13 | PEX opcode |
| 2 | 3 | request length |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 2 | TABLE_INDEX | start |
| 2 | CARD16 | count |

### 7.3.5 Free Lookup Table

5.0

**Name**

**PEXFreeLookupTable**

**Request**

*lut_id*: LOOKUP_TABLE_ID

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

**Description**

This request deletes the association between the lookup table identified by *lut_id* and its resource ID. The lookup table storage will be freed when no other resource references it.

**Encoding: PEXFreeLookupTable**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 6 | PEX opcode |
| 2 | 2 | request length |
| 4 | LOOKUP_TABLE_ID | lookup table ID |

### 7.3.6 Get Defined Indices

5.0

**Name**

**PEXGetDefinedIndices**

**Request**

*lut_id*: LOOKUP_TABLE_ID

**Reply**

*defined_indices*: LISTofTABLE_INDEX

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

**Description**

This request returns in *defined_indices* all the indices currently defined in the lookup table identified by *lut_id*. The entries returned include those predefined and those defined by clients.

**Encoding: PEXGetDefinedIndices**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 9 | PEX opcode |
| 2 | 2 | request length |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | (2*$n$+$p$)/4 | reply length |
| 4 | CARD32 | number of indices returned ($n$) |
| 20 | | unused |
| 2*$n$ | LISTofTABLE_INDEX | defined indices |
| $p$ | | $p$=pad(2*$n$) |

### 7.3.7 Get Table Info

**Name**

**PEXGetTableInfo**

**Request**

| | |
|---|---|
| *drawable_example*: | DRAWABLE_ID |
| *table_type*: | TABLE_TYPE |

**Reply**

| | |
|---|---|
| *definable_entries*: | CARD16 |
| *num_predefined*: | CARD16 |
| *predefined_min*: | TABLE_INDEX |
| *predefined_max*: | TABLE_INDEX |

**Errors**

Drawable
*drawable_example* does not identify a valid drawable.

Match
The drawable identified by *drawable_example* is not supported by the PEX implementation.

Value
*table_type* is not a valid table type.

LookupTable
The requested table type is valid, but not supported by the PEX implementation.

**Description**

This request indicates the entries that are predefined when a table is created and the maximum number of entries it may contain. The values returned apply to tables of the type indicated by *table_type* and for use with drawables of the same depth and class as *drawable_example*. Entries for all indices between and including *predefined_min* and *predefined_max* are predefined for such tables, unless num_predefined is zero, in which case the values of *predefined_min* and *predefined_max* are undefined.

**Encoding: PEXGetTableInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 7 | PEX opcode |
| 2 | 3 | request length |
| 4 | DRAWABLE_ID | example drawable ID |
| 2 | TABLE_TYPE | table type |
| 2 | | unused |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 0 | reply length |
| 2 | | unused |
| 2 | CARD16 | definable entries |
| 2 | CARD16 | number predefined |
| 2 | CARD16 | predefined minimum |
| 2 | CARD16 | predefined maximum |
| 14 | | unused |

### 7.3.8 Get Table Values

**Name**

**PEXGetTableValues**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *lut_id*: | LOOKUP_TABLE_ID |
| *index*: | TABLE_INDEX |
| *value_type*: | TABLE_VALUE_TYPE |
| *fields*: | TABLE_BITMASK |

**Reply**

| | |
|---|---|
| *status*: | TABLE_ENTRY_STATUS |
| *table_type*: | TABLE_TYPE |
| *values*: | LISTofVALUE |

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

Value
1) Index 0 (or 65535) is not valid for this table type.
2) One of the specified fields is illegal.
3) *value_type* does not specify a valid value.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported by the PEX implementation.

**Description**

This request returns the type of the lookup table identified by *lut_id*, and requested values of the entry identified by *index* in that table. If the requested entry is not defined, *status* is given a value of *Default* and the fallback values (§7.1.1) are returned. If the requested entry is defined, then *status* is given a value of *Defined*. One value is returned for each item in *fields*. Floating point values are returned in the floating point format specified in *fp_format*.

If *value_type* is *Set*, the values returned in *values* are those originally specified by the client or predefined by the server. If *value_type* is *Realized*, the values returned are those used after any mapping to a realizable value occurs (§7.1.4).

**Encoding: PEXGetTableValues**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 107 | PEX opcode |
| 2 | 5 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | TABLE_VALUE_TYPE | value type |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 2 | TABLE_INDEX | index |
| 2 | | unused |
| 4 | TABLE_BITMASK | requested fields |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *s*/4 | reply length |
| 2 | TABLE_ENTRY_STATUS | status |
| 2 | TABLE_TYPE | table type |
| 4 | CARD32 | number of values (*n*) |
| 16 | | unused |
| *s*†1 | LISTofVALUE | values |

†1) *s* is the sum of the sizes, in bytes, of all the returned values. Each value selected by *fields* is aligned on a 4-byte boundary.

## 7.4 Requests for Versions 5.0 and 5.1 Compatibility

The requests in this section operate only on table types and fields of those table types which were defined in versions of PEX prior to 5.2. They are superseded by requests added in version 5.2. Table 7-22 shows the old request and its replacement. To maintain backwards compatibility, support of the superseded requests is mandatory for implementations supporting major version 5 of the PEX protocol.

**Table 7-22 . Superseded Lookup Table Requests and Their Replacements**

| Superseded Request | Replacement Request |
|---|---|
| **PEXGetPredefinedEntries** | **PEXGetTableInfo** and **PEXGetTableValues** |
| **PEXGetTableEntries** | **PEXGetTableValues** |
| **PEXGetTableEntry** | **PEXGetTableValues** |
| **PEXSetTableEntries** | **PEXChangeTableValues** |

**PEXGetPredefinedEntries** will generate a *Value* error if called with a table type not supported in 5.0 and 5.1.

**PEXGetTableEntries**, **PEXGetTableEntry** and **PEXSetTableEntries** will generate a *LookupTable* error if called with the resource ID of a table type not supported in 5.0 and 5.1.

### 7.4.1 Get Predefined Entries

5.0

**Name**

**PEXGetPredefinedEntries**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *drawable_example*: | DRAWABLE_ID |
| *table_type*: | TABLE_TYPE_5_1 |
| *start*: | TABLE_INDEX |
| *count*: | CARD16 |

**Reply**

| | |
|---|---|
| *entries*: | LISTofTABLE_ENTRY |

**Errors**

Drawable
*drawable_example* does not identify a valid drawable.

Match
The drawable identified by *drawable_example* is not supported by the PEX implementation.

Value
1) *table_type* is not a valid table type.
2) *start* + *count* is greater than the number of predefined entries.
3) Entry 0 (or 65535) is not valid for this table type.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported by the PEX implementation.

LookupTable
*table_type* is a valid 5.0 or 5.1 table type, but not supported for the specified drawable’s depth and class.

**Description**

This request returns the index and contents of entries that are predefined when a table of *table_type* is created for drawable of the same depth and class as that indicated by *drawable_example*. *Count* entries are returned in *entries*, beginning with the entry identified by *start*.

**Encoding: PEXGetPredefinedEntries**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 8 | PEX opcode |
| 2 | 5 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | DRAWABLE_ID | example drawable ID |
| 2 | TABLE_TYPE_5_1 | table type |
| 2 | TABLE_INDEX | start |
| 2 | CARD16 | count (*n*) |
| 2 | | unused |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *s*/4 | reply length |
| 4 | | unused |
| 4 | CARD32 | number of indices returned (*m*) |
| 16 | | unused |
| *s*†1 | LISTofTABLE_ENTRY | entries |

†1) s is the sum of the sizes of the *n* table entries, in bytes. Each entry may be a different size, depending upon the color type or other information within each entry. Each entry in the list is aligned on a 4-byte boundary.

### 7.4.2 Get Table Entries

5.0

**Name**

**PEXGetTableEntries**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *lut_id*: | LOOKUP_TABLE_ID |
| *start*: | TABLE_INDEX |
| *count*: | CARD16 |
| *value_type*: | TABLE_VALUE_TYPE |

**Reply**

| | |
|---|---|
| *table_type*: | TABLE_TYPE_5_1 |
| *entries*: | LISTofTABLE_ENTRY |

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

Value
1) *start* + *count* exceeds the maximum valid index for this table type.
2) Entry 0 (or 65535) is not valid for this table type.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported by the PEX implementation.

**Description**

This request returns the lookup table type and requested entries from the lookup table identified by *lut_id*. The entries returned are those with indices between and including *start* and *start + count -1*. If an index within that range has no corresponding entry defined, the contents of the fallback entry for the table type is returned for that index. For light tables, entry 1 is returned. Floating point values in *entries* are returned in the floating point format specified in *fp_format*.

If *value_type* is *Set*, the values returned are those originally specified by the client or predefined by the server. If *value_type* is *Realized*, the values returned are those used after any mapping to a realizable value occurs (§7.1.4).

For some table types, this request returns only certain values of an entry. These values are indicated in the table descriptions in §7.2.

**Encoding: PEXGetTableEntries**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 11 | PEX opcode |
| 2 | 4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | TABLE_VALUE_TYPE | value type |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 2 | TABLE_INDEX | start |
| 2 | CARD16 | count (*n*) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *s*/4 | reply length |
| 2 | TABLE_TYPE_5_1 | table type |
| 2 | | unused |
| 4 | CARD32 | number of entries returned (*m*) |
| 16 | | unused |
| *s*†1 | LISTofTABLE_ENTRY | entries |

†1) *s* is the sum of the sizes of the *n* table entries, in bytes. Each entry may be a different size, depending upon the color type or other information within each entry. Each entry in the list is aligned on a 4-byte boundary.

### 7.4.3 Get Table Entry

5.0

**Name**

**PEXGetTableEntry**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *lut_id*: | LOOKUP_TABLE_ID |
| *index*: | TABLE_INDEX |
| *value_type*: | TABLE_VALUE_TYPE |

**Reply**

| | |
|---|---|
| *status*: | TABLE_ENTRY_STATUS |
| *table_type*: | TABLE_TYPE_5_1 |
| *entry*: | TABLE_ENTRY |

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

Value
Entry 0 (or 65535) is not valid for this table type.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported by the PEX implementation.

**Description**

This request returns the lookup table type and the entry identified by *index* from the lookup table identified by *lut_id*. If the requested entry is not defined, the values for the fallback entry are returned and *status* is given a value of *Default*. If the requested entry is defined then *status* is given a value of *Defined*. Floating point values in *entry* are returned in the floating point format specified in *fp_format*.

If *value_type* is *Set*, the values returned in *entry* are those originally specified by the client or predefined by the server. If *value_type* is *Realized*, the values returned are those used after any mapping to a realizable value occurs (§7.1.4).

For some table types, this request returns only certain values of an entry. These values are indicated in the table descriptions in §7.2.

**Encoding: PEXGetTableEntry**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 10 | PEX opcode |
| 2 | 4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | TABLE_VALUE_TYPE | value type |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 2 | TABLE_INDEX | index |
| 2 | | unused |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *s*/4 | reply length |
| 2 | TABLE_ENTRY_STATUS | status |
| 2 | TABLE_TYPE_5_1 | table type |
| 20 | | unused |
| *s*†1 | TABLE_ENTRY | entry |

†1) *s* is the size, in bytes, of the returned entry. The entry is aligned on a 4-byte boundary.

### 7.4.4 Set Table Entries

5.0

**Name**

**PEXSetTableEntries**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *lut_id*: | LOOKUP_TABLE_ID |
| *start*: | TABLE_INDEX |
| *count*: | CARD16 |
| *entries*: | LISTofTABLE_ENTRY |

**Errors**

LookupTable
*lut_id* does not identify a valid lookup table.

Value
1) One of the indices indicated by *start* and *count* is not a valid index for the table type.
2) One of the specified entries contains an illegal value.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported by the PEX implementation.

ColorType
1) The specified color type is not supported.
2) The color type of a color table entry may not be *Indexed*.

Alloc
Defining these entries would exceed the table’s maximum number of entries.

**Description**

This request defines or sets the lookup table entries for all indices between and including *start* and *start* + *count* -1 in the lookup table identified by *lut_id*. If an entry within that range is already defined, then its contents are modified. If an entry is not defined for an index in that range, then this request defines an entry for that index. The values to give the entries are specified in *entries*. There must be *count* entries specified, with each entry in the form required by the type of lookup table. Floating-point values in *entries* are in the floating-point format specified in *fp_format*.

For some table types, this request sets only certain values in an entry. These values are indicated in the table descriptions in §7.2. They are not set by this request, and remain unchanged from their existing value if the entry is already defined. They take on their fallback value if the entry is created by this request.

**Encoding: PEXSetTableEntries**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 12 | PEX opcode |
| 2 | 4+*s*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | LOOKUP_TABLE_ID | lookup table ID |
| 2 | TABLE_INDEX | start |
| 2 | CARD16 | count (*n*) |
| *s*†1 | LISTofTABLE_ENTRY | entries |

†1) *s* is the sum of the sizes of the *n* table entries, in bytes. Each entry may be a different size, depending upon the color type or other information within each entry. The entries in the list are aligned on 4-byte boundaries.

# 8 Name Sets

## 8.1 General

A name set is a PEX resource used to filter primitives during rendering, picking, or searching. They contain names—integers—which are compared against the names associated with a given primitive. The Add Names To Name Set (§ 6.9.1 ) and Remove Names from Name Set (§ 6.9.71 ) output commands associate names with primitives. The name set filters are renderer attributes.

An implementation may limit the number of names a name set may contain. That limit is determined by inquiring *MaxNamesetNames* with request **PEXGetImpDepConstants**.

Name sets are X11 resources and carry all the responsibilities and access rights of X11 resources. The requests used to create and manage name set resources are shown in Table 8-1 .

**Table 8-1 . Name Set Resource Management Requests**

| Request | Description |
|---|---|
| **PEXChangeNameSet** | Modify the name set's names. |
| **PEXCopyNameSet** | Clear the destination name set, then copy the names from the source name set into it. |
| **PEXCreateNameSet** | Create a name set resource. |
| **PEXFreeNameSet** | Free a name set resource. |
| **PEXGetNameSet** | Retrieve the names from a name set. |

## 8.2 Name Set Requests

This section defines the requests that operate on name sets.

### 8.2.1 Change Name Set

5.0

**Name**

**PEXChangeNameSet**

**Request**

| | |
|---|---|
| *ns_id*: | NAME_SET_ID |
| *action*: | {*Replace*, *Add*, *Remove*} |
| *names*: | LISTofNAME |

**Errors**

NameSet
*ns_id* is invalid.

Value
*action* is invalid.

**Description**

This changes the name set identified by *ns_id*. If *action* is *Add*, the names indicated by *names* are added to the name set. If *action* is *Remove*, the names indicated by *names* are removed from the name set. If *action* if *Replace*, all the names in the name set are removed, then the names indicated by *names* are added to the name set. No error is generated if an attempt is made to remove a name not in the name set or to add a name already in the name set.

**Encoding: PEXChangeNameSet**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 53 | PEX opcode |
| 2 | 3+*n* | request length |
| 4 | NAME_SET_ID | name set ID |
| 2 | CARD16 | action |
| | 0 *Add* 5.0 | |
| | 1 *Remove* 5.0 | |
| | 2 *Replace* 5.0 | |
| 2 | | unused |
| 4**n* | LISTofNAME | names (length implicit) |

### 8.2.2 Copy Name Set

5.0

**Name**

**PEXCopyNameSet**

**Request**

| | |
|---|---|
| *src_ns_id*: | NAME_SET_ID |
| *dest_ns_id*: | NAME_SET_ID |

**Errors**

NameSet
*src_ns_id* or *dest_ns_id* is invalid.

**Description**

This requests copies the names in the name set identified by *src_ns_id* to the name set identified by *dest_ns_id*, after first removing all the names from the destination name set.

**Encoding: PEXCopyNameSet**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 50 | PEX opcode |
| 2 | 3 | request length |
| 4 | NAME_SET_ID | source name set ID |
| 4 | NAME_SET_ID | destination name set ID |

### 8.2.3 Create Name Set

5.0

**Name**

**PEXCreateNameSet**

**Request**

*ns_id*: NAME_SET_ID

**Errors**

Alloc
The server failed to allocate the name set.

IDChoice
The resource ID given by *ns_id* is already in use or not in the range assigned to the client.

**Description**

This request creates a name set resource for the specified *ns_id*. The name set is empty when it is created.

**Encoding: PEXCreateNameSet**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 49 | PEX opcode |
| 2 | 2 | request length |
| 4 | NAME_SET_ID | name set ID |

### 8.2.4 Free Name Set

5.0

**Name**

**PEXFreeNameSet**

**Request**

*ns_id*: NAME_SET_ID

**Errors**

NameSet
*ns_id* is invalid.

**Description**

The name set identified by *ns_id* is deleted and its association with that ID removed. The name set is freed when no other resource refers to it.

**Encoding: PEXFreeNameSet**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 51 | PEX opcode |
| 2 | 2 | request length |
| 4 | NAME_SET_ID | name set ID |

### 8.2.5 Get Name Set

5.0

**Name**

**PEXGetNameSet**

**Request**

*ns_id*: NAME_SET_ID

**Reply**

*names*: LISTofNAME

**Errors**

NameSet
*ns_id* is invalid.

**Description**

This request returns the names in the name set identified by *ns_id*.

**Encoding: PEXGetNameSet**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 52 | PEX opcode |
| 2 | 2 | request length |
| 4 | NAME_SET_IDj | name set ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | reply length ($n$) |
| 4 | CARD32 | number of names ($n$) |
| 20 | | unused |
| 4*$n$ | LISTofNAME | names |

# 9 Pipeline Context

## 9.1 General

A pipeline context is a resource used as a receptacle for copies of the renderer's pipeline state (§ 10.1.2). A PEX pipeline context is an X11 resource and carries all the responsibilities and access rights of X11 resources. The contents of a pipeline context resource are copied into the pipeline state during the execution of requests **PEXBeginRendering, PEXRenderNetwork** and **PEXInitRenderer**. A pipeline context can also be used to save and restore, perhaps with modifications, a copy of the pipeline state. This is accomplished with requests **PEXCopyPipelineStateToPC** (§ 10.3.9) and **PEXCopyPCToPipelineState** (§ 10.3.8).

The pipeline state is not a renderer attribute. In many ways it is an implicit component of the renderer. Its fields are not explicitly defined. Instead they are implied by the list of pipeline context fields.

The requests used to create and manage name set resources are shown in Table 9-1 .

**Table 9-1 . Pipeline Context Resource Management Requests**

| Request | Description |
|---|---|
| **PEXChangePipelineContext** | Modify selected attributes of a pipeline context. |
| **PEXCopyPipelineContext** | Copied the specified attributes of the source pipeline context into the destination pipeline context. |
| **PEXCreatePipelineContext** | Create a pipeline context resource, initializing selected attributes. |
| **PEXFreePipelineContext** | Free a pipeline context resource. |
| **PEXGetPipelineContext** | Retrieve selected attributes from a pipeline context. |

### 9.1.1 Item Mask and Item List

Some pipeline context requests have *item_mask* and *item_list* parameters. *item_mask* indicates which values of the pipeline context are to be affected by the request. *item_list* gives those values. The values must be ordered in *item_list* proceeding from the least significant bit (0x00000001) of the first word (0) to the most significant defined bit of the last word (2) of the *item_mask*. Only those values indicated in *item_mask* are to be included in *item_list*.

Table 9-2 alphabetically lists the pipeline context fields in boldface type. For each field, the default value, item mask bit, item list encoding, and a reference to the OC which modifies the field are given. The item mask has words 0 - 2. Its members are also listed, sequentially, in the definition of type PC_BITMASK (§ 3.7.7.98). Each item list encoding is padded to a multiple of 4 bytes. Values which could be encoded in less space are assumed to be placed in the least significant bits of the word. For example, signed values which are padded to 4 bytes are not sign-extended; instead the upper bytes are masked off and unused.

**Table 9-2 . Pipeline Context Fields**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| **active textures (§ 6.9.74)** | | | |
| 2 | CARD16 (*n*) | 0 5.2 | 2/0x00800000 |

**Table 9-2 . Pipeline Context Fields (Continued)**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| 2 | unused | | |
| 2*n* | LISTofTABLE_INDEX | | |
| *p* | pad(2*n*) | | |
| **alpha blend function (§6.9.2)** | | | |
| 4 | ALPHA_BLEND_FUNCTION | *NoBlend* 5.2 | 2/0x00000001 |
| **annotation text alignment (§6.9.3)** | | | |
| 4 | TEXT_ALIGNMENT | {*VAlignNormal, HAlignNormal*} 5.0 | 0/0x00010000 |
| **annotation text height (§6.9.4)** | | | |
| *fp* | FLOAT | 0.01 5.0 | 0/0x00002000 |
| **annotation text path (§6.9.5)** | | | |
| 4 | TEXT_PATH | *Right* 5.0 | 0/0x00008000 |
| **annotation text style (§6.9.6)** | | | |
| 4 | ATEXT_STYLE | *NotConnected* 5.0 | 0/0x00020000 |
| **annotation text up vector (§6.9.7)** | | | |
| 2**fp* | VECTOR_2D | <0.0, 1.0> 5.0 | 0/0x00004000 |
| **aspect source flags†1 (§6.9.39)** | | | |
| 8 | ASF_SPECIFIER_SUBSET (§9.1.2) | {none selected, all *Individual*} 5.0 | 1/0x02000000 |
| **bf active textures (§6.9.12)** | | | |
| 2 | CARD16 (*n*) | 0 5.2 | 2/0x01000000 |
| 2 | unused | | |
| 2*n* | LISTofTABLE_INDEX | | |
| *p* | pad(2*n*) | | |
| **bf data map index (§6.9.8)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00000002 |
| **bf data map method (§6.9.9)** | | | |
| 4+*n*+*p* | DATA_MAP_METHOD_AND_DATA | {*None*, n/a} 5.2 | 2/0x00000004 |
| **bf interior bundle index (§6.9.10)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00000008 |
| **bf interior style (§6.9.15)** | | | |
| 4 | INTERIOR_STYLE | *Hollow* 5.0 | 0/0x80000000 |
| **bf interior style index (§6.9.16)** | | | |
| 4 | TYPE_OR_TABLE_INDEX | 1 5.0 | 1/0x00000001 |
| **bf reflectance bundle index (§6.9.11)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00000010 |
| **bf reflection attributes (§6.9.18)** | | | |
| 4+*sc*+5**fp* | REFLECTION_ATTR | {1.0, 1.0, 1.0, 0.0, 0.0, {*Indexed,* 1}} 5.0 | 1/0x00000004 |
| **bf reflection model (§6.9.19)** | | | |

**Table 9-2 . Pipeline Context Fields (Continued)**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| 4 | REFLECTION_MODEL | *None* 5.0 | 1/0x00000008 |
| **bf surface color (§6.9.13)** | | | |
| 4+*sc* | COLOR_SPECIFIER | {*Indexed,* 1} 5.0 | 1/0x00000002 |
| **bf surface interpolation method (§6.9.17)** | | | |
| 4 | SURFACE_INTERP | *None* 5.0 | 1/0x00000010 |
| **char expansion (§6.9.20)** | | | |
| *fp* | FLOAT | 1.0 5.0 | 0/0x00000040 |
| **char height (§6.9.21)** | | | |
| *fp* | FLOAT | 0.01 5.0 | 0/0x00000200 |
| **char spacing (§6.9.22)** | | | |
| *fp* | FLOAT | 0.0 5.0 | 0/0x00000080 |
| **char up vector (§6.9.23)** | | | |
| 2**fp* | VECTOR_2D | <0.0, 1.0> 5.0 | 0/0x00000400 |
| **color approx index (§6.9.24)** | | | |
| 4 | TABLE_INDEX | 0 5.0 | 1/0x20000000 |
| **condition flags (§6.10.2) (§6.10.12)** | | | |
| 4 | BITMASK | none set 5.2 | 2/0x00000020 |
| **curve approx method (§6.9.25)** | | | |
| 4 | CURVE_APPROX_METHOD | 1 (imp-dep) 5.0 | 0/0x00400000 |
| *fp* | FLOAT | 1.0 5.0 | |
| **data map index (§6.9.26)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00000040 |
| **data map method (§6.9.27)** | | | |
| 4+*n*+*p* | DATA_MAP_METHOD_AND_DATA | {*None,* n/a} 5.2 | 2/0x00000080 |
| **depth cue index (§6.9.28)** | | | |
| 4 | TABLE_INDEX | 0 5.0 | 1/0x01000000 |
| **drawing function (§6.9.29)** | | | |
| 4 | DRAWING_FUNCTION | *GXcopy* 5.2 | 2/0x00000100 |
| **edge bundle index (§6.9.78)** | | | |
| 4 | TABLE_INDEX | 1 5.0 | 1/0x00020000 |
| **facet culling mode (§6.9.30)** | | | |
| 4 | CULL_MODE | *None* 5.0 | 1/0x00000040 |
| **facet distinguish flag (§6.9.31)** | | | |
| 4 | BOOLEAN | *False* 5.0 | 1/0x00000080 |
| **global transform (§6.9.33)-(§6.9.36), (§6.10.3), and (§6.10.6)** | | | |
| 16**fp* | MATRIX | *Identity* 5.0 | 1/0x00080000 |

**Table 9-2 . Pipeline Context Fields (Continued)**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| **global transform hints (§6.9.33)-(§6.9.36), (§6.10.3), and (§6.10.6)** | | | |
| 4 | MATRIX_HINTS | *None* 5.2 | 2/0x00000400 |
| **highlight index (§6.9.37)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00000800 |
| **HLHSR identifier (§6.9.38)** | | | |
| 4 | HLHSR_IDENTIFIER | *Disable* 5.0 | 1/0x08000000 |
| **interior bundle index (§6.9.40)** | | | |
| 4 | TABLE_INDEX | 1 5.0 | 1/0x00001000 |
| **interior style (§6.9.84)** | | | |
| 4 | INTERIOR_STYLE | *Hollow* 5.0 | 0/0x02000000 |
| **interior style index (§6.9.85)** | | | |
| 4 | TYPE_OR_TABLE_INDEX | 1 5.0 | 0/0x04000000 |
| **light source state (§6.9.41)** | | | |
| 4 | CARD32 (*n*) | 0 5.0 | 1/0x00800000 |
| 2**n* | LISTofTABLE_INDEX | | |
| *p* | pad(2**n*) | | |
| **line bundle index (§6.9.42)** | | | |
| 4 | TABLE_INDEX | 1 5.0 | 0/0x01000000 |
| **line cap style (§6.9.43)** | | | |
| 4 | LINE_CAP_STYLE | *Butt* 5.2 | 2/0x00001000 |
| **line color (§6.9.44)** | | | |
| 4+*sc* | COLOR_SPECIFIER | {*Indexed,* 1} 5.0 | 0/0x00200000 |
| **line join style (§6.9.46)** | | | |
| 4 | LINE_JOIN_STYLE | 0 (imp-dep) 5.2 | 2/0x00002000 |
| **line miter limit (§6.9.47)** | | | |
| *fp* | FLOAT | 10.0 5.2 | 2/0x00004000 |
| **line type (§6.9.48)** | | | |
| 4 | LINE_TYPE | *Solid* 5.0 | 0/0x00080000 |
| **line width (§6.9.49)** | | | |
| *fp* | FLOAT | 1.0 5.0 | 0/0x00100000 |
| **local transform (§6.9.50)-(§6.9.53), (§6.10.3), and (§6.10.6)** | | | |
| 16**fp* | MATRIX | *Identity* 5.0 | 1/0x00040000 |
| **local transform hints (§6.9.51) (§6.10.6), (§6.10.3), and (§6.10.6)** | | | |
| 4 | MATRIX_HINTS | *None* 5.2 | 2/0x00008000 |
| **marker bundle index (§6.9.54)** | | | |
| 4 | TABLE_INDEX | 1 5.0 | 0/0x00000008 |
| **marker color (§6.9.55)** | | | |
| 4+*sc* | COLOR_SPECIFIER | {*Indexed*, 1} 5.0 | 0/0x00000004 |

**Table 9-2 . Pipeline Context Fields (Continued)**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| **marker scale factor (§6.9.57)** | | | |
| *fp* | FLOAT | 1.0 5.0 | 0/0x00000002 |
| **marker type (§6.9.58)** | | | |
| 4 | MARKER_TYPE | *Asterisk* 5.0 | 0/0x00000001 |
| **model clip indicator (§6.9.59)** | | | |
| 4 | CARD16 | *NoClip* 5.0 | 1/0x00100000 |
| | 0 *Clip* | | |
| | 1 *NoClip* | | |
| **model clip volume (§6.9.60)** | | | |
| 4 | CARD32 (*n*) | 0 5.0 | 1/0x00200000 |
| *n**6**fp* | LISTofHALF_SPACE | | |
| **more aspect source flags[†1] (§6.9.39)** | | | |
| 16 | ASF_SPECIFIER (§9.1.2) | {none selected, all *Individual*} 5.2 | 2/0x00000200 |
| **name set (§6.9.1) (§6.9.71)** | | | |
| 4 | NAME_SET_ID[†2] | *Null* 5.0 | 1/0x10000000 |
| **parametric surface bundle index (§6.9.62)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00010000 |
| **parametric surface characteristics (§6.9.63)** | | | |
| 4+*n*+*p* | PSURF_CHAR | {*None,* n/a} 5.0 | 1/0x80000000 |
| **pattern reference point (§6.9.65)** | | | |
| 3*fp* | COORD_3D | {0.0, 0.0, 0.0} 5.0 | 1/0x00000200 |
| **pattern reference vector1 (§6.9.65)** | | | |
| 3*fp* | VECTOR_3D | <1.0, 0.0, 0.0> 5.0 | 1/0x00000400 |
| **pattern reference vector2 (§6.9.65)** | | | |
| 3*fp* | VECTOR_3D | <0.0, 1.0, 0.0> 5.0 | 1/0x00000800 |
| **pattern size (§6.9.66)** | | | |
| 2*fp* | VECTOR_2D | <1.0, 1.0> 5.0 | 1/0x00000100 |
| **pick identifier (§6.10.10)** | | | |
| 4 | CARD32 | 0 5.0 | 1/0x04000000 |
| **plane mask (§6.9.67)** | | | |
| 4 | BITMASK | *AllPlanes* 5.2 | 2/0x00020000 |
| **polyline interpolation method (§6.9.68)** | | | |
| 4 | POLYLINE_INTERP | None 5.0 | 0/0x00800000 |
| **primitive antialiasing (§6.9.69)** | | | |
| 4 | PRIM_AA | {implementation-dependent, 0 (imp-dep)} 5.2 | 2/0x00040000 |
| **reflectance bundle index (§6.9.70)** | | | |
| 4 | TABLE_INDEX | 1 5.2 | 2/0x00080000 |
| **reflection attributes (§6.9.87)** | | | |

**Table 9-2 . Pipeline Context Fields (Continued)**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| 4+*sc*+5**fp* | REFLECTION_ATTR | {1.0, 1.0, 1.0, 0.0, 0,0, {*Indexed,* 1}} 5.0 | 0/0x10000000 |
| **reflection model (§6.9.88)** | | | |
| 4 | REFLECTION_MODEL | *None* 5.0 | 0/0x20000000 |
| **rendering color model (§6.9.72)** | | | |
| 4 | RENDERING_COLOR_MODEL | 0 (imp-dep) 5.0 | 1/0x40000000 |
| **sequential processing (§6.10.11)** | | | |
| 4 | BOOLEAN | *True* 5.2 | 2/0x00100000 |
| **surface approximation method (§6.9.75)** | | | |
| 4+2**fp* | SURFACE_APPROX | {1 (imp-dep), 1.0, 1.0} 5.0 | 1/0x00000020 |
| **surface color (§6.9.76)** | | | |
| 4+*sc* | COLOR_SPECIFIER | {*Indexed,* 1} 5.0 | 0/0x08000000 |
| **surface edge color (§6.9.79)** | | | |
| 4+*sc* | COLOR_SPECIFIER | {*Indexed,* 1} 5.0 | 1/0x00010000 |
| **surface edge flag (§6.9.81)** | | | |
| 4 | CARD16 | *Off* 5.0 | 1/0x00002000 |
| | 0 *Off* | | |
| | 1 *On* | | |
| **surface edge type (§6.9.82)** | | | |
| 4 | SURFACE_EDGE_TYPE | *Solid* 5.0 | 1/0x00004000 |
| **surface edge width (§6.9.83)** | | | |
| *fp* | FLOAT | 1.0 5.0 | 1/0x00008000 |
| **surface interpolation method (§6.9.86)** | | | |
| 4 | SURFACE_INTERP | *None* 5.0 | 0/0x40000000 |
| **text alignment (§6.9.89)** | | | |
| 4 | TEXT_ALIGNMENT | {*VAlignNormal, HAlignNormal*} 5.0 | 0/0x00001000 |
| **text bundle index (§6.9.90)** | | | |
| 4 | TABLE_INDEX | 1 5.0 | 0/0x00040000 |
| **text color (§6.9.91)** | | | |
| 4+*sc* | COLOR_SPECIFIER | {*Indexed,* 1} 5.0 | 0/0x00000100 |
| **text font index (§6.9.93)** | | | |
| 4 | TABLE_INDEX | 1 5.0 | 0/0x00000010 |
| **text path (§6.9.94)** | | | |
| 4 | TEXT_PATH | *Right* 5.0 | 0/0x00000800 |
| **text precision (§6.9.95)** | | | |
| 4 | TEXT_PRECISION | *String* 5.0 | 0/0x00000020 |
| **tm perspective correction (§6.9.96)** | | | |
| 4 | TM_PERSPECTIVE_CORRECTION | implementation-dependent 5.2 | 2/0x02000000 |

**Table 9-2 . Pipeline Context Fields (Continued)**

| #of Bytes | Data Type or Value | Default Value | Word/Bit |
|---|---|---|---|
| **tm resource hints (§6.9.97)** | | | |
| 2 | TM_RESOURCE_HINTS | *None* | 2/0x04000000 |
| 2 | CARD16 (*n*) | 0 5.2 | |
| 2**n* | LISTofTABLE_INDEX | | |
| *p* | pad(2**n*) | | |
| **tm sample frequency (§6.9.98)** | | | |
| 4 | TM_SAMPLE_FREQUENCY | *Pixel* 5.2 | 2/0x08000000 |
| **vertex order (§6.9.99)** | | | |
| 4 | VERTEX_ORDER | *Righthand* 5.2 | 2/0x00200000 |
| **view index (§6.9.100)** | | | |
| 4 | TABLE_INDEX | 0 5.0 | 1/0x00400000 |
| **Zbuffer test (§6.9.101)** | | | |
| 4 | Z_TEST | implementation-dependent 5.2 | 2/0x00400000 |

†1) In a pipeline state, all aspect source flags are always enabled. Therefore request **PEXGetPipelineContext** returns the aspect source flags accessed through *aspect source flags* and *more aspect source flags* with all flags enabled.
†2) When a pipeline state is initialized from a pipeline context, the contents of the name set are copied, not the resource identifier of the name set. In a pipeline state, the name set is stored as a list of names. Also, when modifying the *name set* field of the pipeline context, if is permissible to set it to a valid name set resource identifier or **None**, the *Null* resource identifier value defined by X.

### 9.1.2 Aspect Source Flag PC Fields

There is only one set of aspect source flags (ASFs). The *aspect source flags* PC field accesses a subset of the ASFs accessed by the *more aspect source flags* PC field, and for this reason, all clients using version 5.1 or later should use the *more aspect source flags* PC field instead of the *aspect source flags* PC field. The two fields are specified to act together. For example, if an ASF was *Bundled* but was changed to *Individual* through use of the *aspect source flags* PC field, when then inquired through either the *aspect source flags* or *more aspect source flags* field, its returned value, *Individual*, will be the same. The same is true if the value was modified through the *more aspect source flags* field, given the modified ASF is included in the subset of ASFs accessed through the *aspect source flags* field.

**ASF_SPECIFIER** : [enables, asfs : ASF_FIELDS]

**Encoding: ASF_SPECIFIER**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 8 | ASF_FIELDS†1 | aspect source flags selected |
| 8 | ASF_FIELDS†1 | aspect source flags values |

†1) Multiple bits may be set. Each set bit in the *enables* bitmask indicates an ASF whose value is defined by the *asfs* bitmask. A value of 0 in the *asfs* bitmask corresponds to *Bundled*; a value of 1 to *Individual*.

**ASF_SPECIFIER_SUBSET** :
[enables, asfs : ASF_FIELDS_SUBSET]

**Encoding: ASF_SPECIFIER_SUBSET**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | ASF_FIELDS_SUBSET†1 | aspect source flags selected |
| 4 | ASF_FIELDS_SUBSET†1 | aspect source flags values |

†1) Multiple bits may be set. Each set bit in the *enables* bitmask indicates an ASF whose value is defined by the *asfs* bitmask. A value of 0 in the *asfs* bitmask corresponds to *Bundled*; a value of 1 to *Individual*.

**ASF_FIELDS** : BITMASK[2]

**Encoding: ASF_FIELDS**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 4 | ASF_FIELDS_SUBSET | | aspect source flag attributes |
| 4 | BITMASK | | (second word) |
| | 0x00000001 | *data_map_method_ASF* | |
| | 0x00000002 | *bf_data_map_method_ASF* | |
| | 0x00000004 | *line_cap_style_ASF* | |
| | 0x00000008 | *line_join_style_ASF* | |

**ASF_FIELDS_SUBSET** : BITMASK

**Encoding: ASF_FIELDS_SUBSET**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BITMASK | aspect source flag fields |
| | 0x00000001 *marker_type_ASF* 5.0 | |
| | 0x00000002 *marker_scale_ASF* 5.0 | |
| | 0x00000004 *marker_color_ASF* 5.0 | |
| | 0x00000008 *text_font_index_ASF* 5.0 | |
| | 0x00000010 *text_prec_ASF* 5.0 | |
| | 0x00000020 *char_expansion_ASF* 5.0 | |
| | 0x00000040 *char_spacing_ASF* 5.0 | |
| | 0x00000080 *text_color_ASF* 5.0 | |
| | 0x00000100 *line_type_ASF* 5.0 | |
| | 0x00000200 *line_width_ASF* 5.0 | |
| | 0x00000400 *line_color_ASF* 5.0 | |
| | 0x00000800 *curve_approx_ASF* 5.0 | |
| | 0x00001000 *polyline_interp_ASF* 5.0 | |
| | 0x00002000 *interior_style_ASF* 5.0 | |
| | 0x00004000 *interior_style_index_ASF* 5.0 | |
| | 0x00008000 *surface_color_ASF* 5.0 | |
| | 0x00010000 *surface_interp_ASF* 5.0 | |
| | 0x00020000 *reflection_model_ASF* 5.0 | |
| | 0x00040000 *reflection_attr_ASF* 5.0 | |
| | 0x00080000 *bf_interior_style_ASF* 5.0 | |
| | 0x00100000 *bf_interior_style_index_ASF* 5.0 | |
| | 0x00200000 *bf_surface_color_ASF* 5.0 | |
| | 0x00400000 *bf_surface_interp_ASF* 5.0 | |
| | 0x00800000 *bf_reflection_model_ASF* 5.0 | |
| | 0x01000000 *bf_reflection_attr_ASF* 5.0 | |
| | 0x02000000 *surface_approx_ASF* 5.0 | |
| | 0x04000000 *surface_edges_ASF* 5.0 | |
| | 0x08000000 *surface_edge_type_ASF* 5.0 | |
| | 0x10000000 *surface_edge_width_ASF* 5.0 | |
| | 0x20000000 *surface_edge_color_ASF* 5.0 | |
| | 0x40000000 *parametric_surface_characteristics_ASF* 5.2 | |
| | 0x80000000 reserved | |

## 9.2 Pipeline Context Requests

The requests that operate on pipeline contexts are defined in this section.

### 9.2.1 Change Pipeline Context

5.0

**Name**

**PEXChangePipelineContext**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pc_id*: | PIPELINE_CONTEXT_ID |
| *item_mask*: | PC_BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

PipelineContext
The pipeline context identified by *pc_id* does not exist.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Value
A non-defined bit in *item_mask* is set or a value in *item_list* is invalid.

ColorType
A value in *item_list* requests an unsupported color type.

NameSet
The name set specified in *item_list* is not defined.

**Description**

This request sets the values indicated by *item_mask* in the pipeline context identified by *pc_id*. The *item_list* contains the new values, listed in the order defined in §9.1.1. Floating point values in *item_list* are in the floating point format specified by *fp_format*.

**Encoding: PEXChangePipelineContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 18 | PEX opcode |
| 2 | 6+*n*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | PIPELINE_CONTEXT_ID | pipeline context identifier |
| 12 | PC_BITMASK | item mask (See Table 9-2 or §3.7.7.98 for the bit assignments.) |
| *n* | LISTofVALUE | item list (See Table 9-2 for the item encodings.) |

### 9.2.2 Copy Pipeline Context

5.0

**Name**

**PEXCopyPipelineContext**

**Request**

| | |
|---|---|
| *src_pc_id*: | PIPELINE_CONTEXT_ID |
| *dest_pc_id*: | PIPELINE_CONTEXT_ID |
| *item_mask*: | PC_BITMASK |

**Errors**

PipelineContext
The pipeline context identified by *src_pc_id* or *dest_pc_id* does not exist.

Value
A non-defined bit in *item_mask* is set.

**Description**

This request copies values in the pipeline context identified by *src_pc_id* to the pipeline context identified by *dest_pc_id*. The destination pipeline context must already exist. Only the values indicated by *item_mask* are copied.

**Encoding: PEXCopyPipelineContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 15 | PEX opcode |
| 2 | 6 | request length |
| 4 | PIPELINE_CONTEXT_ID | source pipeline context identifier |
| 4 | PIPELINE_CONTEXT_ID | destination pipeline context identifier |
| 12 | PC_BITMASK | item mask (See Table 9-2 or §3.7.7.98 for the bit assignments.) |

### 9.2.3 Create Pipeline Context

**Name**

**PEXCreatePipelineContext**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pc_id*: | PIPELINE_CONTEXT_ID |
| *item_mask*: | PC_BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

IDChoice
The resource ID given in *pc_id* is already in use or not in the range assigned to the client.

Value
A non-defined bit in *item_mask* is set or a value in *item_list* is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

ColorType
A value in *item_list* requests an unsupported color type.

Alloc
The server failed to allocate the pipeline context.

NameSet
The name set specified in *item_list* is not defined.

**Description**

This request creates a pipeline context resource for the specified *pc_id*. The *item_mask* indicates those pipeline context values to be explicitly set when the pipeline context is created. The *item_list* contains those values, listed in the order defined in §9.1.1. Floating point values in *item_list* are in the floating point format specified in *fp_format*.

**Encoding: PEXCreatePipelineContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 14 | PEX opcode |
| 2 | 6+*n*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | PIPELINE_CONTEXT_ID | pipeline context identifier |
| 12 | PC_BITMASK | item mask (See Table 9-2 or §3.7.7.98 for the bit assignments.) |
| *n* | LISTofVALUE | item list (See Table 9-2 for the item encodings.) |

### 9.2.4 Free Pipeline Context

5.0

**Name**

**PEXFreePipelineContext**

**Request**

*pc_id*: PIPELINE_CONTEXT_ID

**Errors**

PipelineContext
The pipeline context identified by *pc_id* does not exist.

**Description**

This request deletes the association between the resource ID and the pipeline context. The pipeline context storage is freed when no other resource references it.

**Encoding: PEXFreePipelineContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 16 | PEX opcode |
| 2 | 2 | request length |
| 4 | PIPELINE_CONTEXT_ID | pipeline context identifier |

### 9.2.5 Get Pipeline Context

5.0

**Name**

**PEXGetPipelineContext**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pc_id*: | PIPELINE_CONTEXT_ID |
| *item_mask*: | PC_BITMASK |

**Reply**

| | |
|---|---|
| *item_list*: | LISTofVALUE |

**Errors**

PipelineContext
The pipeline context identified by *pc_id* does not exist.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Value
A non-defined bit in *item_mask* is set.

**Description**

This request returns the values indicated by *item_mask* from the pipeline context identified by *pc_id*. The values are returned in *item_list*, in the order in which they’re defined in §9.1.1. Floating-point values are returned in the floating-point format specified in *fp_format*.

When the *name set* value of a pipeline context refers to a name set resource that has been freed, the value *AlreadyFreed* (§3.7.7.120) is returned for the name set ID.

**Encoding: PEXGetPipelineContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 17 | PEX opcode |
| 2 | 6 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | PIPELINE_CONTEXT_ID | pipeline context identifier |
| 12 | PC_BITMASK | item mask (See Table 9-2 or §3.7.7.98 for the bit assignments.) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *n*/4 | reply length |
| 24 | | unused |
| *n* | LISTofVALUE | item list (See Table 9-2 for the item encodings.) |

# 10 Renderers

## 10.1 General

A PEX renderer is the resource responsible for drawing output primitives and applying their attributes. A PEX renderer is an X11 resource and carries all the responsibilities and access rights of X11 resources. Associated with the renderer are other resources: a pipeline context giving the initial values of the pipeline state, lookup tables holding values for many of the parameters used during rendering, and name sets identifying primitives to be highlighted, made invisible, or made detectable during picking. During rendering, the renderer is associated with one drawable, to which all output is directed.

### 10.1.1 Rendering Controls

The requests that create and manage the renderer resource are listed in Table 10-2 . Those that initiate and control rendering are listed in Table 10-2 . Rendering starts when a **PEXBeginRendering** request associates the renderer with a drawable. A **PEXEndRendering** request disassociates the renderer from the drawable. Immediate mode rendering, where output commands are sent directly to a renderer from the client to be rendered immediately to the drawable, is accomplished primarily by **PEXRenderOutputCommands** requests. Structure mode rendering, where output commands are earlier stored in structures and later traversed by a renderer, is accomplished either by **PEXRenderNetwork** requests or **PEXRenderOutputCommands** requests with Execute Structure, Conditional Execute Structure, Include Structure, and Conditional Include Structure output commands. **PEXRenderNetwork** implicitly does Begin- and EndRendering. **PEXRenderElements** can also be used to render a subset of elements in a structure.

A renderer is said to be “rendering” when its *renderer state* attribute (§ 10.2.36) is *Rendering* or *MultipassRendering*.

**Table 10-1 . Renderer Resource Management Requests**

| Request | Description |
|---|---|
| **PEXCreateRenderer** | Create a renderer resource and optionally set attributes to non-default values. |
| **PEXChangeRenderer** | Change renderer attribute values. |
| **PEXGetRendererAttributes** | Retrieve renderer attribute values. |
| **PEXFreeRenderer** | Free a renderer resource. |

**Table 10-2 . Renderer Control Requests**

| Request | Description |
|---|---|
| **PEXBeginRendering** | Associate a drawable with a renderer and initialize the renderer. |
| **PEXBindDrawableToRenderer** | Flush any pending output to the drawable. Associate a different drawable with the renderer without initializing the renderer. |
| **PEXRenderOutputCommands** | Render a list of output commands. |
| **PEXRenderElements** | Render a range of output commands stored in a structure. |
| **PEXInitRenderer** | Initialize the renderer’s pipeline state to values in renderer’s pipeline context. |
| **PEXClearRenderer** | Clear the drawable and associated ancillary buffers. |
| **PEXFlushRenderer** | Render any pending output to the drawable. |

**Table 10-2 . Renderer Control Requests (Continued)**

| Request | Description |
|---|---|
| **PEXBeginStructure** | Update current path, save a copy of pipeline state and set model transformation. |
| **PEXEndStructure** | Update current path and restore last saved copy of pipeline state. |
| **PEXAccumulateState** | Traverse a structure network and update pipeline state without rendering. |
| **PEXCopyPipelineStateToPC** | Save a copy of renderer’s pipeline state in an independent pipeline context. |
| **PEXCopyPCToPipelineState** | Restore renderer’s pipeline state to values in an independent pipeline context. |
| **PEXInitMultipass** | Initialize the renderer for multipass rendering. |
| **PEXNextPass** | Demarcate the end of a multipass rendering pass and return status. |
| **PEXNextPassWithoutReply** | Demarcate the end of a multipass rendering pass and do not return status. |
| **PEXGetZBuffer** | Get the content of the Z-buffer in device-independent format. |
| **PEXPutZBuffer** | Write the content of the Z-buffer. |
| **PEXCopyZBufferToPixmap** | Copy the content of the Z-buffer to a drawable of equal depth. |
| **PEXCopyPixmapToZBuffer** | Copy the content of a drawable of equal depth to the Z-buffer. |
| **PEXCopyAlphaBufferToPixmap** | Copy the content of the alpha buffer to a drawable of equal depth. |
| **PEXCopyPixmapToAlphaBuffer** | Copy the content of a drawable of equal depth to the alpha buffer. |
| **PEXEndRendering** | Optionally flush any pending output to the drawable. Disassociate the renderer and the drawable. Set the renderer’s *renderer state* attribute to *Idle*. |
| **PEXRenderNetwork** | Render a structure network. Implicitly perform a **PEXBeginRendering** and **PEXEndRendering**. |

### 10.1.1.1 Multi-pass Rendering

Multipass rendering is provided so that a renderer can perform operations such as multipass transparency that require the output commands defining a single scene to be processed more than once. The operations performed during multipass rendering and the effects desired are indicated by the pipeline state and the renderer attributes.

A multipass session is initiated by a **PEXInitMultipass** request. All output commands subsequently passed to the renderer are considered part of a pass, many of which may be required to produce the requested effects. The first pass consists of the output commands sent to the renderer between the **PEXInitMultipass** request and a **PEXNextPass** or **PEXNextPassWithoutReply** request. These latter two requests demarcate the end of the current pass and the beginning of the next. The server expects the same output commands to be sent in each pass.

One pass after another is performed until no more are needed. The number of passes required is determined by the server and conveyed to the client through the reply of **PEXNextPass**. That request returns a value indicating the number of additional passes needed, if the server can determine that, or simply that more passes are needed or that no more are needed.

A client performing simple multipass rendering begins a scene by sending a **PEXInitMultipass** request. It then continually sends passes, demarcated by **PEXNextPass** requests, until **PEXNextPass** returns zero for the number of additional passes required. Options are defined for aborting a multipass session and for informing the server to produce the best image it can with only one more pass.

Whether a server is capable of multipass rendering is determined by corresponding support for capabilities that require multipass rendering. The only defined capabilities in PEX 5.2 that require multipass rendering are transparency methods *TransparencyTwoPass and TransparencyMultipass*.

### 10.1.2 Pipeline State

The pipeline state is a data structure that maintains the values that can be modified by output commands during rendering or picking. It is created when the renderer is associated with a drawable by **PEXBeginRendering**, **PEXBeginPickOne**, or **PEXBeginPickAll**. It exists until the renderer receives the corresponding **PEXEndRendering**, **PEXEndPickOne**, or **PEXEndPickAll** request. The pipeline state also exists for the duration of requests **PEXRenderNetwork**, **PEXPickOne**, and **PEXPickAll**.

The pipeline state contains the same information as a pipeline context, and is initialized by the pipeline context associated with the renderer during **PEXBeginRendering**, **PEXBeginPickOne**, **PEXBeginPickAll**, **PEXRenderNetwork**, **PEXPickOne**, **PEXPickAll**, and **PEXInitRenderer** requests.

Values in the pipeline state are modified by output commands as well as **PEXCopyPCtoPipelineState**. The description of each output command (See Chapter 6) defines which pipeline state value the output command modifies. When a primitive output command is drawn by the renderer, the attribute values currently in effect in the pipeline state are applied to the primitive.

### 10.1.3 Renderer Interaction with Ancillary Buffers

A renderer manages the Z-buffer needed for some hidden line/hidden surface algorithms. Whenever an implicit or explicit **PEXBeginRendering** occurs and the renderer's *HLHSR_mode* indicates that Z-buffering is to be performed or if the renderer's *clear Z* attribute is *True*, a Z-buffer suitable for use with the specified drawable is allocated, if necessary, and bound to the drawable. (A new Z-buffer is only allocated if a Z-buffer does not exist for the drawable or if the Z-buffer which is bound to the drawable is not suitable, e.g. a resize has occurred and the Z-buffer has not yet been resized.) If the renderer's *HLHSR_mode* does not indicate that Z-buffering is to be performed and the *clear Z* attribute is *False* at the time **PEXBeginRendering** is processed, but the mode is later changed by a **PEXChangeRenderer** request to indicate that Z-buffering is to be performed, the Z-buffer is allocated, if necessary, at that time. A Z-buffer is also allocated, if necessary, when a **PEXClearRenderer**, **PEXGetZBuffer**, **PEXPutZBuffer**, **PEXCopyZBufferToPixmap**, or **PEXCopyPixmapToZBuffer** request is made when the renderer's state is not *Idle.*

If the renderer's *clear Z* attribute is *True* when **PEXBeginRendering** is processed or a new Z-buffer is allocated, then the Z-buffer is cleared to the value in the renderer's *clear Z value* only in the region specified by the renderer's clip list.

A renderer also manages the buffer used for alpha blending and certain methods of transparency. This buffer is known as an alpha buffer as it contains the resulting alpha values from certain blending operations. Whenever an implicit or explicit **PEXBeginRendering** occurs and the renderer's *clear alpha* attribute is *True* or the renderer's *transparency method* indicates an alpha buffer is needed, an alpha buffer is allocated and bound to the drawable. If the *transparency method* does not indicate an alpha buffer is needed when **PEXBeginRendering** is processed, but the method is later changed by a **PEXChangeRenderer** request to a method which does require an alpha buffer, the alpha buffer is allocated, if necessary, at that time. Also, if the transparency method is *TransparencyAlphaBlend*, the alpha buffer will not be allocated, if necessary, until the pipeline state's *alpha blend function* is set to a value which requires an alpha buffer (i.e. a function which uses destination alpha as an input). An alpha buffer is also allocated, if necessary, when a **PEXClearRenderer**, **PEXCopyAlphaBufferToPixmap**, or **PEXCopyPixmapToAlphaBuffer** request is made when the renderer's state is not *Idle.*

If the renderer's *clear alpha* attribute is *True* when **PEXBeginRendering** is processed or a new alpha buffer is allocated, the alpha buffer is cleared to the alpha value in the renderer's *background color*. If no alpha value is provided in the renderer's *background color*, the default value of 1.0 is used. The alpha buffer is cleared only in the region specified by the renderer's clip list.

If the Z-buffer or alpha buffer could not be allocated, an *Alloc* error is generated and rendering is aborted; the *renderer state* is set to *Idle*.

The Z-buffer and alpha buffer are not deallocated when the *renderer state* becomes *Idle*. They remain bound to the drawable until the drawable is freed. When the renderer state next becomes *Rendering* or *Picking*, an attempt is made to reuse them. If they cannot be reused (for instance, if the drawable has been resized and the Z-buffer has not been resized to match), they are reallocated as necessary.

Accumulation buffers are also ancillary buffers. However accumulation buffers interact directly with the drawable rather than with a renderer. See Chapter 15 for more information on accumulation buffers.

Request **PEXGetAncillaryBufferInfo** (§4.1.1) returns the ancillary buffers, and their characteristics, supported for given visuals and/or pixmap depths.

### 10.1.4 Response to Drawable Changes

If the drawable associated with a renderer is destroyed or resized while the renderer state is *Rendering*, an implicit **PEXEndRendering** is performed by the server with a *flush* parameter of *False* in order to return the renderer to the *Idle* state. All subsequent output and traversal commands (all the requests listed in Table 10-2 except **PEXBeginRendering** and **PEXEndRendering**) are ignored. If the renderer state is *Picking* or *MultipassRendering*, those operations are similarly aborted. No errors are generated; however, at a subsequent **PEXBeginRendering**, an *Alloc* error may be generated if an attempt to reallocate ancillary buffers fails.

If the drawable associated with a renderer is moved, exposed, or occluded while the renderer state is not *Idle*, the server continues processing subsequent requests using the newly-modified drawable.

The impact of drawable changes on accumulation buffers is described in Chapter 15.

### 10.1.5 Item Masks and Item Lists

Some requests use *item_mask* and *item_list* parameters. *item_mask* indicates which renderer attributes are to be affected by the request. *item_list* gives those values. The values must be ordered in *item_list* in the same order as listed in §10.2. Only those values indicated in *item_mask* are to be included in *item_list*. The item list encodings and the bit assignments for *item_mask* are given in §10.2. The most significant bit (0x80000000) of each BITMASK is reserved to indicate additional words for the item mask. The additional words will immediately follow each preceding BITMASK in the encoding.

## 10.2 Renderer Attributes

The renderer attributes are listed in Table 10-3 and described fully below. They are set with **PEXChangeRenderer** requests, and can also be specified when a renderer is created with a **PEXCreateRenderer** request.

Unless otherwise indicated in its detailed description below, each renderer attribute is a member of the Global attribute set, or the renderer attribute is not a member of any attribute set because of its lack of effect on primitive rendering. The attribute sets are defined in §6.5. The definition of each attribute set includes a list of the primitives affected by the attributes in the set.

If a renderer attempts to read a value from a lookup table whose resource ID is *Null*, the fallback attributes for the lookup table (as listed in §7.2) are used except as noted in the individual descriptions of the lookup tables below. If a renderer attempts to read a value from a name set whose resource ID is *Null*, the result will be as if the name set exists but is empty. If the renderer's pipeline context attribute contains the resource ID *Null*, the default values for the pipeline context (§9.1) are copied to the renderer's pipeline state during a **PEXBeginRendering**, **PEXRenderNetwork** or **PEXInitRenderer** request. Note, when modifying any renderer attribute whose type is a resource identifier, it is permissible to set that attribute to either an appropriate valid resource identifier or to **None**, the *Null* resource identifier value defined by X.

**Table 10-3 . Renderer Attributes**

| Name | Data Type | Default Value | Item Mask Word/Bit |
|---|---|---|---|
| pipeline context (§10.2.34) | PIPELINE_CONTEXT_ID | *Null* 5.0 | 0/0x00000001 |
| current path (§10.2.9) | LISTofELEMENT_REF | *Null* 5.0 | 0/0x00000002 |

**Table 10-3 . Renderer Attributes (Continued)**

| Name | Data Type | Default Value | Item Mask Word/Bit |
|---|---|---|---|
| marker bundle (§ 10.2.26) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000004 |
| text bundle (§ 10.2.37) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000008 |
| line bundle (§ 10.2.25) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000010 |
| interior bundle (§ 10.2.21) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000020 |
| edge bundle (§ 10.2.15) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000040 |
| view table (§ 10.2.43) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000080 |
| color table (§ 10.2.8) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000100 |
| depth cue table (§ 10.2.11) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000200 |
| light table (§ 10.2.24) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000400 |
| color approx table (§ 10.2.7) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00000800 |
| pattern table (§ 10.2.29) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00001000 |
| text font table (§ 10.2.38) | LOOKUP_TABLE_ID | *Null* 5.0 | 0/0x00002000 |
| highlight inclusion (§ 10.2.18) | NAME_SET_ID | *Null* 5.0 | 0/0x00004000 |
| highlight exclusion (§ 10.2.17) | NAME_SET_ID | *Null* 5.0 | 0/0x00008000 |
| invisibility inclusion (§ 10.2.23) | NAME_SET_ID | *Null* 5.0 | 0/0x00010000 |
| invisibility exclusion (§ 10.2.22) | NAME_SET_ID | *Null* 5.0 | 0/0x00020000 |
| renderer state (§ 10.2.36) | CARD16 | *Idle* 5.0 | 0/0x00040000 |
| HLHSR mode (§ 10.2.20) | HLHSR_MODE | *Off* 5.0 | 0/0x00080000 |
| NPC subvolume (§ 10.2.28) | NPC_SUBVOLUME | {(0.0,0.0,0.0),(1.0,1.0,1.0)} 5.0 | 0/0x00100000 |
| viewport (§ 10.2.42) | VIEWPORT | {imp. dep., imp. dep., *True*} 5.0 | 0/0x00200000 |
| clip list (§ 10.2.6) | LISTofDEVICE_RECT | *Null* 5.0 | 0/0x00400000 |
| pick inclusion (§ 10.2.32) | NAME_SET_ID | *Null* 5.1 | 0/0x00800000 |
| pick exclusion (§ 10.2.31) | NAME_SET_ID | *Null* 5.1 | 0/0x01000000 |
| pick start path (§ 10.2.33) | LISTofELEMENT_REF | *Null* 5.1 | 0/0x02000000 |
| background color (§ 10.2.2) | COLOR_SPECIFIER | {*Indexed*, 0} 5.1 | 0/0x04000000 |
| clear I (§ 10.2.4) | BOOLEAN | *False* 5.1 | 0/0x08000000 |
| clear Z (§ 10.2.5) | BOOLEAN | *True* 5.1 | 0/0x10000000 |
| echo mode (§ 10.2.14) | ECHO_MODE | *NoEcho* 5.1 | 0/0x20000000 |
| clear Z value (§ 10.2.5) | FLOAT | 0.0 5.2 | 0/0x40000000 |
| clear alpha (§ 10.2.3) | BOOLEAN | *False* 5.2 | 1/0x00000001 |
| PHIGS interior bundle mode (§ 10.2.30) | BOOLEAN | *False* 5.2 | 1/0x00000002 |
| extent size table (§ 10.2.16) | LOOKUP_TABLE_ID | *Null* 5.2 | 1/0x00000004 |
| data mapping table (§ 10.2.10) | LOOKUP_TABLE_ID | *Null* 5.2 | 1/0x00000008 |
| highlight table (§ 10.2.19) | LOOKUP_TABLE_ID | *Null* 5.2 | 1/0x00000010 |
| marker glyph table (§ 10.2.27) | LOOKUP_TABLE_ID | *Null* 5.2 | 1/0x00000020 |
| transparency method (§ 10.2.40) | ENUM_TYPE_INDEX | implementation-dependent 5.2 | 1/0x00000040 |

**Table 10-3 . Renderer Attributes (Continued)**

| Name | Data Type | Default Value | Item Mask Word/Bit |
|---|---|---|---|
| transparency thresholds (§ 10.2.41) | LISTofFLOAT | {1.0} | 1/0x00000080 |
| pixel color mode (§ 10.2.35) | SWITCH | *Off* | 1/0x00000100 |
| echo color (§ 10.2.12) | COLOR_SPECIFIER | implementation-dependent | 1/0x00000200 |
| echo method (§ 10.2.13) | ENUM_TYPE_INDEX | implementation-dependent | 1/0x00000400 |
| texture mapping table (§ 10.2.39) | LOOKUP_TABLE_ID | *Null* | 1/0x00000800 |

### 10.2.1 Renderer Attribute Encodings

The renderer attribute encodings are given where the attribute is described in the sections below. Each item list encoding is padded to a multiple of 4 bytes. Values which could be encoded in less space are assumed to be placed in the least significant bits of the word. Those attributes consisting of only a resource ID have the following encoding:

**Encoding: Resource ID Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | resource ID |

### 10.2.2 Background Color

The color to use when clearing the image buffer of the drawable associated with the renderer. The color approximation indicated by the pipeline state's *color approximation index* is applied to this color when the image buffer is cleared, and only the area identified by the renderer's *clip list* is given the background color.

The alpha component of the background color defines the value to use when clearing the alpha buffer of the drawable associated with the renderer. If the background color does not have an alpha component, a value of 1.0 is used. Again, only the area defined by the renderer's *clip list* is cleared with this alpha value.

**Encoding: Background Color Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4+*sc* | COLOR_SPECIFIER | background color (see COLOR for *sc*) |

### 10.2.3 Clear Alpha

Indicates whether the alpha buffer, if one is bound to the current drawable, is cleared during **PEXBeginRendering** and **PEXRenderNetwork** requests. If *clear alpha* is *True*, the alpha buffer is cleared to the alpha value specified in the renderer's *background color*, otherwise it is not. If no alpha value is specified by the background color, a default value of 1.0 is used. Only pixels within the renderer's clip list are cleared. If *clear alpha* is *True*, and if the drawable associated with the renderer has the capability to support an alpha buffer, an alpha buffer will be allocated, if necessary (see §10.1.3).

**Encoding: Clear Alpha Buffer Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BOOLEAN | clear alpha buffer |

### 10.2.4 Clear Image

Indicates whether the image buffer of the drawable associated with the renderer is cleared to the renderer's *background color* during **PEXBeginRendering** and **PEXRenderNetwork** requests. If *clear image* is *True*, the image buffer is cleared, otherwise it is not. Only pixels within the renderer's *clip list* are modified during the clear.

**Encoding: Clear Image Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BOOLEAN | clear image |

### 10.2.5 Clear Z-buffer and Clear Z-buffer Value

*clear Z* indicates whether the Z-buffer is cleared during **PEXBeginRendering** and **PEXRenderNetwork** requests. If *clear Z* is *True*, the Z-buffer is cleared to the value specified in the renderer's *clear Z value*, otherwise it is not. The *clear Z value* is in the range [0.0, 1.0] in DC space (§2.2.1.4). Only pixels within the renderer's clip list are cleared. If *clear Z* is *True*, and if the drawable associated with the renderer has the capability to support a Z-buffer (as indicated by **PEXGetEnumeratedTypeInfo**), a Z-buffer will be allocated, if necessary (see §10.1.3).

**Encoding: Clear Z-buffer Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BOOLEAN | clear Z-buffer |

**Encoding: Clear Z-buffer Value Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | clear Z-buffer value |

### 10.2.6 Clip List

A list of rectangles in device coordinates (§2.2.1.4) that define the areas of the drawable in which rendering may occur. If the list is *Null*, the entire drawable is eligible. If the list is not *Null*, rendering occurs only in those pixels that are within the rectangles specified in the list. The rectangles are bounded by the pixels addressed by device rectangle parameters *xmin, xmax, ymin,* and *ymax*, inclusive. The pixels addressed by *xmin, xmax, ymin*, and *ymax* are considered part of the clipping rectangle. The result is undefined if any of the rectangles in the clip list overlap.

**Encoding: Clip List Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of rectangles (*n*) |
| 8**n* | LISTofDEVICE_RECT | rectangles |

### 10.2.7 Color Approximation Table

Identifies the color approximation table (§7.2.3) from which to draw the color approximation parameters indicated by the pipeline state’s *color approximation index*.

### 10.2.8 Color Table

Identifies the color table (§7.2.2) used to resolve indexed colors.

### 10.2.9 Current Path

Maintains the structure-network path formed by rendering requests. The path is initialized by **PEXBeginRendering**, **PEXRenderNetwork** and **PEXInitRenderer** to a single element reference with a structure identifier and an offset of zero. The structure identifier is set to zero for **PEXBeginRendering** and **PEXInitRenderer**, and to the identifier of the network’s root structure for **PEXRenderNetwork**.

This attribute cannot be set explicitly by the client. If an attempt is made to modify this attribute via **PEXChangeRenderer**, the item_mask bit and corresponding item_list value will be ignored. It can only be inquired.

**Encoding: Current Path Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of element references in path (*n*) |
| 8**n* | LISTofELEMENT_REF | path |

### 10.2.10 Data Mapping Table

Identifies the data mapping table (§7.2.4) from which to draw the data mapping method indicated by the pipeline state’s *data mapping index* and *data mapping backface index* and corresponding ASFs.

Data Mapping Table is a member of the Data Mapping attribute set.

### 10.2.11 Depth Cue Table

Identifies the depth cue table (§7.2.5) from which to draw the depth cueing parameters indicated by the pipeline state’s *depth cue index*.

### 10.2.12 Echo Color

Provides the color used by some echo methods when the *echo mode* renderer attribute is *Echo* or *UnEcho*. The echo method is determined by the *echo method* renderer attribute.

To support clients using PEX versions prior to 5.2, *echo color* can also be modified with escape *SetEchoColor*.

**Encoding: Echo Color Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4+*sc* | COLOR_SPECIFIER | echo color |

### 10.2.13 Echo Method

Indicates how primitives are to be echoed and unechoed. The registered echo methods are listed in Table 10-4 .

**Table 10-4 . Enumerated Type *EchoMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *DestructiveColor* | Echo a primitive by drawing it in the color defined by the *echo color* renderer attribute. This color overrides the primitive's color. Lighting, shading, depth cueing, transparency, texture mapping, and data mapping are not applied to the primitive. The color is applied to both the front-facing and back-facing facets of surface primitives.<br>This method is destructive, meaning the image rendered in echo mode *Echo* cannot be redrawn correctly in echo mode *UnEcho*. The image must be cleared and rerendered to be restored. |
| 2 | *NonDestructiveColor* | This method is identical to method *DestructiveColor* except the image is redrawn correctly in echo mode *UnEcho*. |
| 3 | *XORColor* | Echo a primitive by performing the logical XOR operation on the pixel values in the frame buffer that the primitive modifies with the pixel value obtained by applying the renderer's current color approximation to the renderer's *echo color* renderer attribute. The resulting pixels are stored back in the frame buffer, overriding the primitive's color. Lighting, shading, depth cueing, transparency, texture mapping, and data mapping are not applied to the primitive. The color is applied to both the front-facing and back-facing facets of surface primitives.<br>The pipeline state's *drawing function* is forced to behave like function *GXcopy* when echo method *XORColor* is enabled.<br>This method is non-destructive, meaning the image rendered in echo mode *Echo* is redrawn correctly in echo mode *UnEcho*. |

Whether primitives are to be echoed is controlled by the *echo mode* renderer attribute.

**Encoding: Echo Method Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | ENUM_TYPE_INDEX | echo method |

### 10.2.14 Echo Mode

Indicates whether primitives are echoed, unechoed or drawn normally when passed to a renderer. The method used for echoing and unechoing is identified by the *echo method* renderer attribute.

If a primitive is designated to be both highlighted and echoed, it is echoed and not highlighted.

**Encoding: Echo Mode Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | ECHO_MODE | echo mode |

### 10.2.15 Edge Bundle Table

Identifies the edge bundle table (§7.2.6) used to resolve bundled edge attributes.

Edge Bundle Table is a member of the Edge attribute set.

### 10.2.16 Extent Size Table

Identifies the extent size table (§7.2.7) from which to fetch the extent size used with the Test Extent and Test Extent 2D output commands.

### 10.2.17 Highlight Exclusion Name Set

Identifies a name set used in determining which primitives to highlight (§10.2.18). Primitives with at least one name in this name set are not highlighted.

### 10.2.18 Highlight Inclusion Name Set

Identifies a name set used in determining which primitives to highlight when they are rendered. Those primitives are highlighted that have one of its names, as given by the pipeline state's *name set*, in the renderer's *highlight inclusion name set* when the primitive is drawn, and none of its names in the renderer's *highlight exclusion name set*.

### 10.2.19 Highlight Table

Identifies the highlight table (§7.2.8) from which to fetch the parameters which describe how to highlight primitives. An entry of the highlight table is selected by the pipeline state's *highlight index*. If no highlight table is associated with the renderer, the method used for highlighting is implementation-dependent.

### 10.2.20 HLHSR Mode

The hidden line/hidden surface method used when resolving visibility of overlapping primitives. The modes defined by PEX are described in Table 10-5 .

**Table 10-5 . Enumerated Type: *HLHSRMode***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Off* | All output primitives are drawn in the order they are processed. No attempt is made to remove hidden surfaces. 5.0 |
| 2 | *ZBuffer* | Visibility is resolved at each pixel using a depth-, or Z-buffering technique. The Z-buffering method and the number of bits of precision in the Z values is drawable-dependent. 5.0 |
| 3 | *Painters* | Output primitives are buffered as they are processed. When an “end rendering” occurs with flush = *True*, the primitives in the buffer are sorted based on the average depth and rendered back-to-front. 5.0 |
| 4 | *Scanline* | Output primitives are buffered as they are received. When an “end rendering” occurs with flush = *True*, the primitives in the buffer are sorted and visibility is computed in scan line order. 5.0 |
| 5 | *HiddenLineOnly* | Only visible lines will be drawn. Output primitives may be buffered as they are received. When an “end rendering” occurs with flush = *True*, the primitives in the buffer are sorted and hidden line computation is performed. 5.0 |
| 6 | *ZBufferId* | The same method as for *ZBuffer*, but the pipeline state’s *HLHSR identifier* determines which primitives HLHSR applies to. See §6.9.38. 5.1 |

**Encoding: HLHSR Mode Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | ENUM_TYPE_INDEX | HLHSR mode |

### 10.2.21 Interior Bundle Table

Identifies the interior bundle table (§7.2.9) used to resolve bundled interior attributes.

Interior Bundle Table is a member of the General Surface, Styled and Shaded Surface, Curved Surface, and Parametric Surface attribute sets.

### 10.2.22 Invisibility Exclusion Name Set

Identifies a name set used in determining which primitives are invisible (§10.2.23). Primitives with at least one name in this name set cannot be made invisible.

### 10.2.23 Invisibility Inclusion Name Set

Identifies a name set used in determining which primitives are to be invisible—not drawn—when passed to a renderer. Those primitives are invisible that have one of their names, as given by the pipeline state’s *name set*, in the renderer’s *invisibility inclusion name set* when the primitive is passed to the renderer, and none of their names in the renderer’s *invisibility exclusion name set*.

### 10.2.24 Light Table

Identifies the light table (§7.2.10) from which to draw light source definitions indicated by the pipeline state’s *light source state*.

Light Table is a member of the General Surface attribute set.

### 10.2.25 Line Bundle Table

Identifies the line bundle table (§7.2.11)used to resolve bundled line attributes.

Line Bundle Table is a member of the Line, Shaded Line, and Curve attribute sets.

### 10.2.26 Marker Bundle Table

Identifies the marker bundle table (§7.2.12) used to resolve bundled marker attributes.

Marker Bundle Table is a member of the Marker attribute set.

### 10.2.27 Marker Glyph Table

Identifies the marker glyph table (§7.2.13) from which to draw the definition of markers selected by the pipeline state’s *marker type* respectively. If no marker glyph table is associated with the renderer, the pipeline state’s *marker type* selects one of the registered marker types as defined in Table 6-32 (§6.9.58).

### 10.2.28 NPC Subvolume

The portion of normalized project coordinates that is to be mapped to the renderer’s *viewport* (§10.2.42).

**Encoding: NPC Subvolume Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 6**fp* | NPC_SUBVOLUME | NPC subvolume |

### 10.2.29 Pattern Table

Identifies the pattern table (§7.2.14) from which to draw pattern definitions.

Pattern Table is a member of the Styled and Shaded Surface attribute set.

### 10.2.30 PHIGS Interior Bundle Mode

This attribute is provided to support the interior bundle organization of PHIGS PLUS (§7.2.9).

PHIGS Interior Bundle Mode is a member of the General Surface, Styled and Shaded Surface, Curved Surface, and Parametric Surface attribute sets.

**Encoding: PHIGS Interior Bundle Mode**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | BOOLEAN | mode |

### 10.2.31 Pick Exclusion Name Set

Identifies a name set used in determining which primitives are eligible for picking (§ 10.2.32). A primitive with at least one name in this name set cannot be picked.

### 10.2.32 Pick Inclusion Name Set

Identifies a name set used in determining which primitives are detectable during picking. Those primitives are detectable that have one of their names, as given by the pipeline state's *name set*, in the renderer's *pick inclusion name set* when the primitive is passed to the renderer, and none of their names in the renderer's *pick exclusion name set*. Other criteria described in Chapter 11 are also used to determine detectable primitives.

### 10.2.33 Pick Start Path

If the server cannot return all of the selected primitives, the *pick start path* is set by requests **PEXPickAll** and **PEXEndPickAll** to allow the client to continue selecting primitives by invoking request **PEXPickAll** or **PEXBeginPickAll** again. The *pick start path* identifies the position within a structure network after which to begin the next pick request. The search begins at the structure element immediately following the one indicated by the *pick start path*.

**Encoding: Pick Start Path Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of entries in path ($n$) |
| 8*$n$ | LISTofELEMENT_REF | path |

### 10.2.34 Pipeline Context

The requests **PEXBeginRendering**, **PEXInitRenderer**, and **PEXRenderNetwork** initialize all values in the renderer's pipeline state to those values in the pipeline context indicated by this attribute.

### 10.2.35 Pixel Color Mode

This attribute is provided to support color defined as pixel values.

When the pixel color mode is *On*, if a color type is *Indexed*, the index value is interpreted as a pixel value. The color lookup table is therefore not used to resolve *Indexed* colors. Colors of all other types are converted to pixel values by finding their closest match in the colormap of the drawable which is bound to the renderer. Table 10-6 describes the values which override the given pipeline state and renderer attributes when the pixel color mode is *On*.

**Table 10-6 Behaviors Enabled by the Pixel Color Mode**

| Attribute | Behavior |
|---|---|
| Pipeline state’s *data map method ASF* | *Individual* |
| Pipeline state’s *data map method* | *None* |
| Pipeline state’s *bf data map method ASF* | *Individual* |
| Pipeline state’s *bf data map method* | *None* |
| Pipeline state’s *rendering color model* | ignored (rendering is done with pixels) |
| Pipeline state’s *vertex order* | ignored (no reflectance calculations) |
| Pipeline state’s *polyline interpolation method ASF* | *Individual* |
| Pipeline state’s *polyline interpolation method* | *None* |
| Pipeline state’s *surface interpolation method ASF* | *Individual* |
| Pipeline state’s *surface interpolation method* | *None* |
| Pipeline state’s *light source state* | empty |
| Pipeline state’s *active textures* | empty |
| Pipeline state’s *bf active textures* | empty |
| Renderer’s *depth cue table* entry’s *mode* | *Disabled* |
| Renderer’s *transparency method* | *None* |
| Pipeline state’s *primitive antialiasing* | no primitive types enabled |
| Renderer’s *color table* | ignored (indexed colors are pixel values) |
| Renderer’s *color approximation table* | ignored (color approximation is unnecessary) |

Fetch *PixelColorSupported* with request **PEXGetImpDepConstants**, to determine if the pixel color mode is supported.

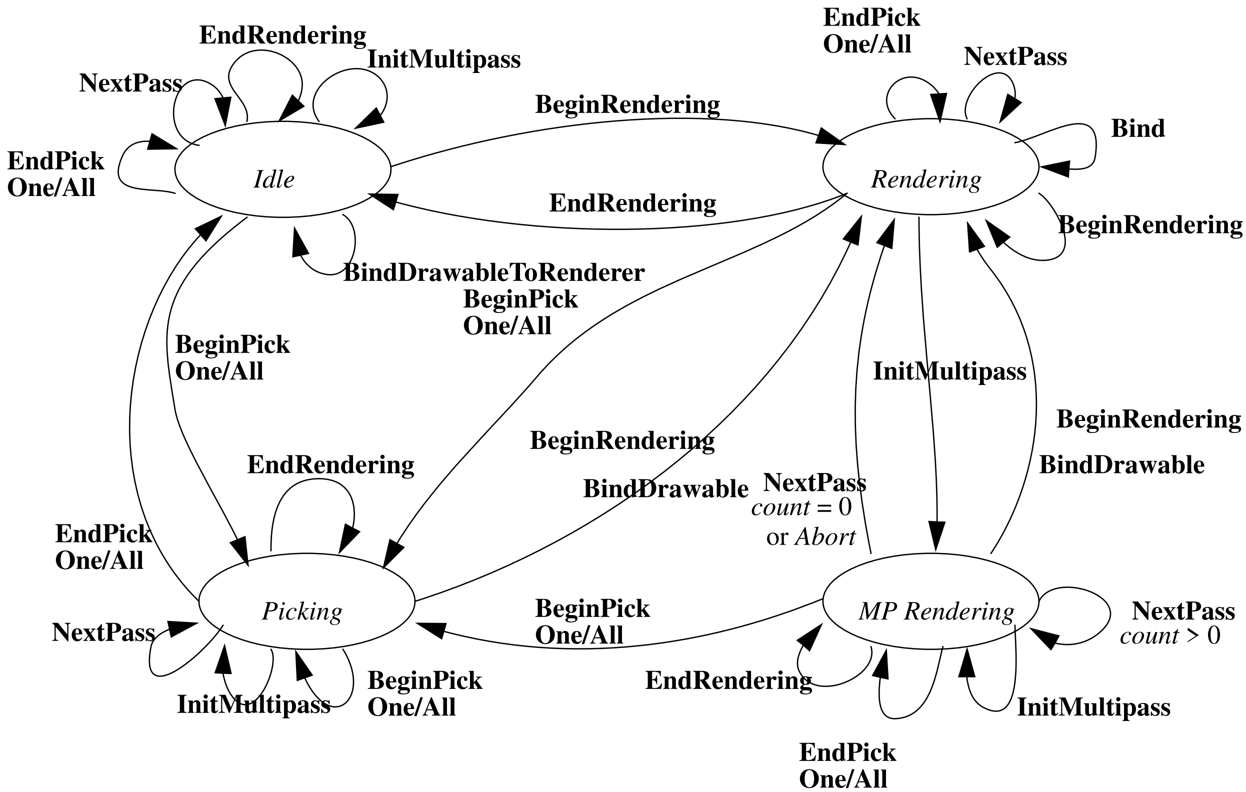
**Encoding: Pixel Color Mode**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | SWITCH | mode |

### 10.2.36 Renderer State

Holds the current state of the renderer, either *Idle*, *Rendering*, *Picking* or *MultipassRendering*. The state is set to *Rendering* by a **PEXBeginRendering** request, and is subsequently set to *Idle* by a **PEXEndRendering** request or certain error conditions (described in the requests that potentially raise those conditions). It also has the value *Rendering* while a **PEXRenderNetwork** request is in progress. The renderer state is set to *Picking* by **PEXBeginPickOne** and **PEXBeginPickAll** requests, and subsequently to *Idle* by **PEXEndPickOne** or **PEXEndPickAll** requests or certain error conditions. It also has the value *Picking* during **PEXPickOne** or **PEXPickAll** requests. It is set to *MultipassRendering* by **PEXInitMultipass**, and subsequently to *Rendering* by the final **PEXNextPass** or **PEXNextPassWithoutReply** which completes the multipass rendering. The renderer state transitions are shown in the diagram below:

## Renderer State Transitions




This attribute cannot be set explicitly by the client. If an attempt is made to modify this attribute via **PEXChangeRenderer**, the *item_mask* bit and corresponding *item_list* value will be ignored. It can only be inquired.

**Encoding: Renderer State Renderer Attribute**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 4 | CARD16 | | renderer state |
| | 0 | *Idle* 5.0 | |
| | 1 | *Rendering* 5.0 | |
| | 2 | *Picking* 5.1 | |
| | 3 | *MultipassRendering* 5.2 | |

### 10.2.37 Text Bundle Table

Identifies the text bundle table (§7.2.15) used to resolve bundled text attributes.

Text Bundle Table is a member of the Annotation and Text attribute set.

### 10.2.38 Text Font Table

Identifies the text font table (§7.2.16) from which to draw the font group indicated by the pipeline state's *text font index*.

Text Font Table is a member of the Annotation and Text attribute set.

### 10.2.39 Texture Mapping Table

Identifies the texture mapping table (§7.2.17) from which to fetch texture map definitions indicated by the pipeline state’s *active textures* and *bf active textures*.

Texture Mapping Table is a member of the Styled and Shaded Surface attribute set.

### 10.2.40 Transparency Method

Indicates the method used to perform transparency. The alpha component of the primitive’s color is used for transparency computations. If the primitive does not define the alpha value, the default value is determined differently depending on the primitive type. For surface primitives, the default alpha value is 1.0 - *transmission coefficient*. The transmission coefficient is part of the reflection attributes (§6.9.87). When the pipeline state’s *facet distinguish flag* is *True*, the alpha value for back-facing facets is 1.0 - *bf transmission coefficient*, where the back-facing transmission coefficient is defined by the pipeline state’s *bf reflection attributes* (§6.9.18). For all other, non-surface primitives, the default alpha value is 1.0.

**Table 10-7 . Enumerated Type: *TransparencyMethod***

| Method | Value | Description |
|---|---|---|
| *None* | 0 | No transparency. |
| *ScreenDoor* | 1 | Screen-door transparency. |
| *AlphaBlend* | 2 | Alpha blending. If the server supports an alpha buffer, this mode can be used to perform transparency without multipass rendering. Otherwise, the client will have to sort primitives for reasonable results. The blending function used is determined by the Alpha Blend Function OC (§6.9.2) |
| *TwoPass* | 3 | Render only those primitives or portions of primitives with an alpha value of 1.0 in the first pass; render (blend) all other primitives in the second pass. |
| *Multipass* | 4 | Meaningful for multipass rendering only; same as *transparency method None* if multipass rendering not active. The number of passes and the pixels drawn during each pass are controlled by the *transparency thresholds* renderer attribute. |

**Encoding: Transparency Method**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | ENUM_TYPE_INDEX | transparency method |

### 10.2.41 Transparency Thresholds

Defines the thresholds used by transparency method *Multipass*. The *transparency method* is also a renderer attribute. In the first pass, all pixels with alpha values greater or equal to the first threshold are rendered. In the *i*th pass, where *i* is in the range [2..*n*], all pixels with alpha values less than the *i-1*th threshold value and greater than or equal to the *i*th threshold value are rendered. In the *n+1*th pass, all pixels with alpha values less than the *n*th threshold and greater than or equal to 0.0 are rendered. There are *n+1* passes for *n* threshold values.

**Encoding: Transparency Thresholds**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | number of multipass transparency thresholds (*n*) |
| *n**fp* | LISTofFLOAT | list of thresholds |

### 10.2.42 Viewport

The area within a drawable in which the rendered image appears, specified in device coordinates (DC—§2.2.1.4). A viewport whose boundaries are outside of the drawable may be specified. Geometry that is mapped to values outside the drawable is clipped.

When the viewport is bound to the renderer, if the viewport's *use_drawable* flag is *True* the viewport width and height are obtained from the drawable's current width and height, and the viewport's depth is that indicated by the *viewport* attribute. If the *use_drawable* flag is *False*, the viewport width, height and depth are all taken from the *viewport* attribute. The viewport is then set to the largest rectangle, justified to the lower left corner of the drawable, that achieves an isotropic mapping (§2.2.1) to the renderer's *NPC subvolume*. The viewport remains in the same position relative to the lower left hand corner of the drawable after a resize event occurs.

**Encoding: Viewport Renderer Attribute**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 12+2**fp* | VIEWPORT | viewport |

### 10.2.43 View Table

Identifies the view table (§7.2.18) from which to draw the view definition indicated by the pipeline state's *view index*.

## 10.3 Renderer Requests

This section defines the requests that operate on renderers. Some requests defined by PEX versions 5.0 and 5.1 have been superseded. Those requests are defined in §10.4, and their replacements are identified there.

### 10.3.1 Accumulate State

5.1

**Name**

**PEXAccumulateState**

**Request**

*rdr_id*: RENDERER_ID
*path*: LISTofELEMENT_REF

**Errors**

Renderer
*rdr_id* is invalid.

StructurePermission
An invalid position in a locked structure (§ 12.1.1.3) is given in *path*.

Path
*path* contains the ID of a nonexistent structure or an invalid element position.

RendererState
The renderer state is *MultipassRendering*.

**Description**

If the *renderer state* of the renderer identified by *rdr_id* is *Rendering* or *Picking*, this request traverses to the structure element indicated by *path* and modifies only the pipeline state according to the output commands in that path. No primitives are rendered. The initial pipeline state is that in effect at the time the request. If the *renderer_state* is *Idle*, this request has no effect.

During the traversal, the *offset* member of the last entry in the renderer's *current path* is incremented by one for each structure element traversed, including those that have no effect on the pipeline state. No *structure* member of the current path is changed. No action beyond incrementing the current path offset is taken when Execute Structure, Conditional Execute Structure, or Conditional Return elements are traversed during this request.

This request is not allowed while the renderer is performing multipass rendering.

**Encoding: PEXAccumulateState**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 98 | PEX opcode |
| 2 | 3+2**n* | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | CARD32 | number of element references in path (*n*) |
| 8**n* | LISTofELEMENT_REF | path |

### 10.3.2 Begin Rendering

5.0

**Name**

**PEXBeginRendering**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |

**Errors**

Renderer
*rdr_id* is invalid.

Drawable
*drawable_id* is invalid.

Match
1) The renderer was not created for drawables of the same root, depth, and visual ID as the specified drawable.
2) The drawable identified by *drawable_id* is of an unsupported type.

Alloc
The server was unable to allocate the necessary resources.

RendererState
The renderer's state is *Rendering*, *Picking* or *MultipassRendering*.

**Description**

This request associates the drawable identified by *drawable_id* with the renderer identified by *rdr_id*. The output of the renderer is directed to the indicated drawable until the renderer receives a **PEXEndRendering** or **PEXBindDrawableToRenderer** request. The renderer's pipeline state is initialized by copying to it the values in the pipeline context associated with the renderer, or, if no pipeline context is associated, the default pipeline context values. The renderer's *rendering state* is set to *Rendering*. Its *current path* is initialized to one entry whose *structure* and *offset* are 0. If the renderer's *clear image* attribute is *True*, the drawable's image buffer is initialized to the renderer's *background color* within the areas identified by the renderer's *clip list*. If the renderer's *clear Z* attribute is *True*, then the Z-buffer is cleared to the value in the renderer's *clear Z value* within the areas identified by the renderer's *clip list* (§10.1.3). If the renderer's *clear alpha* attribute is *True*, the alpha buffer is cleared to the value specified by the renderer's *background color* (§10.1.3).

If the renderer's renderer state is *Rendering*, *Picking* or *MultipassRendering*, the operation is aborted and this request performs the above operations but also generates an error. The new renderer state is *Rendering*.

**Encoding: PEXBeginRendering**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 24 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |

### 10.3.3 Begin Structure

5.0

**Name**

**PEXBeginStructure**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *s_id*: | CARD32 |

**Errors**

Renderer
*rdr_id* is invalid.

**Description**

If the *renderer state* of the renderer identified by *rdr_id* is *Rendering*, *Picking* or *MultipassRendering*, this request causes the renderer to save the current values of its pipeline state such that they can subsequently be restored by a **PEXEnd-Structure** request. The *offset* of the last entry in the renderer's *current path* is incremented by one, and a new entry is added to the path. The structure member of that entry is given the value *s_id*; the offset member is given the value 0.

After saving the pipeline state, the pipeline state's *global transform* is set to the concatenation of the current *local transform* and the current *global transform*. The *local transform* is then set to the identity.

If the *renderer state* is *Idle*, this request has no effect.

**Encoding: PEXBeginStructure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 26 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | CARD32 | structure ID |

### 10.3.4 Bind Drawable to Renderer

**Name**

**PEXBindDrawableToRenderer**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |

**Errors**

Renderer
*rdr_id* is invalid.

Drawable
*drawable_id* is invalid.

Match
1) The renderer was not created for drawables of the same root, depth, and visual ID as the specified drawable.
2) The drawable identified by *drawable_id* is of an unsupported type.

Alloc
The server was unable to allocate the necessary resources.

RendererState
The renderer's state is *MultipassRendering* or *Picking*.

**Description**

This request associates the drawable identified by *drawable_id* with the renderer identified by *rdr_id*. The output of the renderer is directed to the indicated drawable until the renderer receives another **PEXBindDrawableToRenderer** request or a **PEXEndRendering** request.

If the renderer's *renderer state* is *Idle*, this request has no effect. Otherwise the operation in progress is aborted by an implicit **PEXFlushRenderer** (§10.3.16) with the flush parameter set to *True*. The renderer's *renderer state* is set to *Rendering*. If the *renderer state* is initially *MultipassRendering* or *Picking*, a *RendererState* error is additionally generated.

The renderer's pipeline state, attributes (except the *renderer state*), and the drawable's image and ancillary buffers are not affected by this request.

**Encoding: PEXBindDrawableToRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 132 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |

### 10.3.5 Change Renderer

5.0

**Name**

**PEXChangeRenderer**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *rdr_id*: | RENDERER_ID |
| *item_mask*: | BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

Renderer
*rdr_id* is invalid.

Value
1) A value in *item_list* is invalid.
2) An invalid bit is set in *item_mask*.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

PipelineContext
The pipeline context ID given in *item_list* is invalid.

NameSet
A name set ID given in *item_list* is invalid.

LookupTable
A lookup table ID given in *item_list* is invalid.

Match
A lookup table was not created for drawables of the same root, depth, and visual ID as the renderer.

**Description**

This request changes renderer attributes of the renderer identified by *rdr_id*. The *item_mask* and *item_list* specify which components to change (§10.1.5). Floating-point values in *item_list* are in the floating-point format specified in *fp_format*.

**Encoding: PEXChangeRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 21 | PEX opcode |
| 2 | 4+(*s*+*n*)/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 4 | BITMASK | item mask |
| s[†1] | BITMASK | item mask second word 5.2 |
| n | LISTofVALUE | item list |

†1) s = ((0x80000000 & item mask) ? 4 : 0)

### 10.3.6 Clear Renderer

**Name**

**PEXClearRenderer**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *clear_control*: | BITMASK |

**Errors**

Renderer
*rdr_id* is invalid.

Value
An invalid bit in *clear_control* is set.

Alloc
The server was unable to allocate a Z-buffer or an alpha buffer.

**Description**

If the *renderer state* of the renderer identified by *rdr_id* is *Rendering*, *Picking*, or *MultipassRendering*, this request clears the buffers of the drawable associated with that renderer. If bit 0 (0x00000001) of *clear_control* is 1, the renderer's *background color* is drawn in the drawable's image buffer in the regions indicated by the renderer's *clip list* (§10.2.4). If bit 1 (0x00000002) is 1, the Z-buffer, if any, is cleared (§10.2.5). If bit 2 (0x00000004) is 1, and there is an alpha buffer associated with the renderer's drawable, the alpha buffer is cleared to the value specified by the renderer's *background color* (§10.2.3). See also §10.1.3.

If the renderer's rendering state is *Idle*, this request has no effect.

**Encoding: PEXClearRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 108 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | BITMASK | clear control |
| | 0x00000001 clear-image bit 5.2 | |
| | 0x00000002 clear-Z-buffer bit 5.2 | |
| | 0x00000004 clear-alpha-buffer bit 5.2 | |

### 10.3.7 Copy Alpha Buffer to Pixmap

**Name**

**PEXCopyAlphaBufferToPixmap**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *pixmap*: | PIXMAP_ID |

**Errors**

Renderer
*rdr_id* is invalid.

Pixmap
*pixmap* is invalid.

Match
1) The drawable associated with the renderer cannot have an alpha buffer.
2) The pixmap's depth is not equal to or greater than the depth of the alpha buffer bound to the drawable associated with the renderer.
3) The pixmap's width or height do not match those of the drawable associated with the renderer.

Alloc
The server was unable to allocate an alpha buffer.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is *Rendering* or *MultipassRendering,* the content of the alpha buffer bound to the drawable associated with the renderer is copied to the specified *pixmap*. It is undefined what is copied for obscured regions of the drawable to which the alpha buffer is bound. The renderer's *clip list* is not considered by this request. The format of the values copied to the pixmap is implementation-dependent.

The pixmap must be at least as deep as the alpha buffer. If the pixmap is deeper than the alpha buffer, the values will be copied into the pixmap in an implementation-dependent way. To determine the depth and implementation-dependent type and range of the alpha buffer values, use request **PEXGetAncillaryBufferInfo** (§4.1.1).

The validity of the values copied if the *renderer state* is *Picking* is implementation-dependent. If the renderer's state is *Idle*, this request has no effect.

**Encoding: PEXCopyAlphaBufferToPixmap**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 118 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | PIXMAP_ID | pixmap ID |

### 10.3.8 Copy PC to Pipeline State

**Name**

**PEXCopyPCToPipelineState**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *pc_id*: | PIPELINE_CONTEXT_ID |
| *item_mask*: | PC_BITMASK |

**Errors**

Renderer
*rdr_id* is invalid.

PipelineContext
*pc_id* is invalid.

Value
An invalid bit is set in *item_mask*.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is not *Idle*, selected fields of the pipeline context identified by *pc_id* are copied to the renderer's pipeline state. The fields are selected by *item_mask* (§9.1.1).

If the *renderer state* is *Idle* when this request is invoked, the pipeline state is not modified.

**Encoding: PEXCopyPCToPipelineState**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 121 | PEX opcode |
| 2 | 6 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | PIPELINE_CONTEXT_ID | pipeline context ID |
| 12 | PC_BITMASK | item mask |

### 10.3.9 Copy Pipeline State to PC

**Name**

**PEXCopyPipelineStateToPC**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *pc_id*: | PIPELINE_CONTEXT_ID |
| *item_mask*: | PC_BITMASK |

**Errors**

Renderer
*rdr_id* is invalid.

PipelineContext
*pc_id* is invalid.

Value
An invalid bit is set in *item_mask*.

NameSet
The pipeline context's name set resource ID is invalid.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is not *Idle*, selected fields of the renderer's pipeline state are copied to the pipeline context identified by *pc_id*. The fields are selected by *item_mask* (§9.1.1).

If the item mask indicates that the pipeline state's current name set is to be copied, the pipeline context must contain the resource identifier of a valid name set resource into which the pipeline state's current name set values will be copied. Any names existing in the name set resource prior to the copy operation will be deleted. If the name set resource ID is invalid, a *Value* error will be generated.

If the *renderer state* is *Idle* when this request is invoked, the pipeline context is not modified.

**Encoding: PEXCopyPipelineStateToPC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 120 | PEX opcode |
| 2 | 6 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | PIPELINE_CONTEXT_ID | pipeline context ID |
| 12 | PC_BITMASK | item mask |

### 10.3.10 Copy Pixmap to Alpha Buffer

**Name**

**PEXCopyPixmapToAlphaBuffer**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *pixmap*: | PIXMAP_ID |

**Errors**

Renderer
*rdr_id* is invalid.

Pixmap
*pixmap* is invalid.

Match
1) The drawable associated with the renderer cannot have an alpha buffer.
2) The pixmap’s depth is not equal to or greater than the depth of the alpha buffer bound to the drawable associated with the renderer.
3) The pixmap’s width or height do not match those of the drawable associated with the renderer.

Alloc
The server was unable to allocate an alpha buffer.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is *Rendering* or *MultipassRendering,* the content of the specified *pixmap* is copied to the alpha buffer bound to the drawable associated with the renderer. Locations in the alpha buffer corresponding to obscured regions of the drawable associated with the renderer are not modified. The renderer’s *clip list* is not considered by this request.

The format of the values copied from the pixmap is implementation-dependent. The results are undefined if the pixmap is deeper than the alpha buffer and the depth of the alpha buffer used to originally load the pixmap, with request **PEXCopyAlphaBufferToPixmap**, differs from the depth of the alpha buffer this request modifies.

The values copied will have an implementation-dependent effect if the *renderer state* is *Picking*. If the *renderer state* is *Idle*, this request has no effect.

**Encoding: PEXCopyPixmapToAlphaBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 119 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | PIXMAP_ID | pixmap ID |

### 10.3.11 Copy Pixmap to Z-buffer

**Name**

**PEXCopyPixmapToZBuffer**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *pixmap*: | PIXMAP_ID |

**Errors**

Renderer
*rdr_id* is invalid.

Pixmap
*pixmap* is invalid.

Match
1) The drawable associated with the renderer cannot have a Z-buffer.
2) The pixmap's depth is not equal to or greater than the depth of the Z-buffer bound to the drawable associated with the renderer.
3) The pixmap's width or height do not match those of the drawable associated with the renderer.

Alloc
The server was unable to allocate a Z-buffer.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is *Rendering* or *MultipassRendering,* the content of the specified *pixmap* is copied to the Z-buffer bound to the drawable associated with the renderer. Locations in the Z-buffer corresponding to obscured regions of the drawable associated with the renderer are not modified. The renderer's *clip list* is not considered by this request.

The format of the values copied from the pixmap is implementation-dependent. The results are undefined if the pixmap is deeper than the Z-buffer and the depth of the Z-buffer used to originally load the pixmap, with request **PEXCopyZBufferToPixmap**, differs from the depth of the Z-buffer this request modifies.

The values copied will have an implementation-dependent effect if the *renderer state* is *Picking*. If the *renderer state* is *Idle*, this request has no effect.

**Encoding: PEXCopyPixmapToZBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 117 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | PIXMAP_ID | pixmap ID |

### 10.3.12 Copy Z-buffer to Pixmap

5.2

**Name**

**PEXCopyZBufferToPixmap**

**Request**

*rdr_id*: RENDERER_ID
*pixmap*: PIXMAP_ID

**Errors**

Renderer
*rdr_id* is invalid.

Pixmap
*pixmap* is invalid.

Match
1) The drawable associated with the renderer cannot have a Z-buffer.
2) The pixmap’s depth is not equal to or greater than the depth of the Z-buffer bound to the drawable associated with the renderer.
3) The pixmap’s width or height do not match those of the drawable associated with the renderer.

Alloc
The server was unable to allocate a Z-buffer.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is *Rendering* or *MultipassRendering,* the content of the Z-buffer bound to the drawable associated with the renderer is copied to the specified *pixmap*. It is undefined what is copied for obscured regions of the drawable to which the Z-buffer is bound. The renderer’s *clip list* is not considered by this request. The format of the values copied to the pixmap is implementation-dependent.

The pixmap must be at least as deep as the Z-buffer. If the pixmap is deeper than the Z-buffer, the values will be copied into the pixmap in an implementation-dependent way. To determine the depth and implementation-dependent type and range of the Z-buffer values, use request **PEXGetAncillaryBufferInfo** (§4.1.1).

The validity of the values copied if the *renderer state* is *Picking* is implementation-dependent. If the renderer’s state is *Idle*, this request has no effect.

**Encoding: PEXCopyZBufferToPixmap**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 116 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | PIXMAP_ID | pixmap ID |

### 10.3.13 Create Renderer

**Name**

**PEXCreateRenderer**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *rdr_id*: | RENDERER_ID |
| *drawable_example*: | DRAWABLE_ID |
| *item_mask*: | BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

IDChoice
The resource ID given in *rdr_id* is already in use or not in the range assigned to the client.

Drawable
*drawable_example* is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Match
1) The drawable identified by *drawable_example* is of an unsupported type.
2) One of the specified lookup table resources was not created for the same root, depth, and visual ID as the drawable.

Value
1) One of the specified values in *item_list* is invalid.
2) An invalid bit is set in *item_mask*.

PipelineContext
The pipeline context ID given in *item_list* is invalid.

NameSet
A name set ID given in *item_list* is invalid.

LookupTable
A lookup table ID given in *item_list* is invalid.

Alloc
The server failed to allocate the renderer.

**Description**

This request creates a renderer resource for the specified *rdr_id*. The renderer can be used to render to drawables with the same root, depth, and visual ID as the drawable specified by *drawable_example*. *item_mask* and *item_list* specify those renderer attributes to be set when the renderer is created (§ 10.1.5). Floating point values in *item_list* will be in the floating point format specified in *fp_format*.

**Encoding: PEXCreateRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 19 | PEX opcode |
| 2 | 5+($s$+$n$)/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | example drawable ID |
| 4 | BITMASK | item mask |
| $s$ †1 | BITMASK | item mask second word |
| $n$ | LISTofVALUE | item list |

†1) $s$ = ((0x80000000 & item mask) ? 4 : 0)

### 10.3.14 End Rendering

5.0

**Name**

**PEXEndRendering**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *flush*: | BOOLEAN |

**Errors**

Renderer
*rdr_id* is invalid.

RendererState
The renderer's state is *Picking* or *MultipassRendering*.

**Description**

This requests terminates rendering by the specified renderer to the associated drawable. If *flush* is *True*, all pending output is rendered to the drawable prior to terminating rendering. If *flush* is *False*, all pending output is discarded. The *renderer state* is set to *Idle*.

If the *renderer state* is already *Idle* when this request is received, this request has no effect. If the *renderer state* is *Picking* or *MultipassRendering* when this request is received, an error is generated, but this request otherwise has no effect.

**Encoding: PEXEndRendering**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 25 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 1 | BOOLEAN | flush |
| 3 | | unused |

### 10.3.15 End Structure

5.0

**Name**

**PEXEndStructure**

**Request**

*rdr_id*: RENDERER_ID

**Errors**

Renderer
*rdr_id* is invalid.

RendererState
No matching **PEXBeginStructure** request was received prior to this request.

**Description**

If the *renderer state* of the renderer specified by *rdr_id* is *Rendering*, *Picking* or *MultipassRendering*, this request replaces the values in the renderer's current pipeline state with those of the last-saved copy of the pipeline state (saved during the most recent **PEXBeginStructure** request). The last entry of the renderer's *current path* is removed, and subsequent output commands cause the *offset* of the new last entry of *current path* to be incremented. This request does not cause the new last entry to be incremented.

If the *renderer state* is *Idle*, this request has no effect.

**Encoding: PEXEndStructure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 27 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |

### 10.3.16 Flush Renderer

**Name**

**PEXFlushRenderer**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *flush*: | BOOLEAN |

**Errors**

Renderer
*rdr_id* is invalid.

**Description**

If *flush* is *True*, all pending output is rendered onto the drawable associated with the renderer, or if the renderer state is *Picking*, all pending output is processed. If *flush* is *False*, all pending output is discarded.

If the *renderer state* is *Idle*, this request has no effect.

**Encoding: PEXFlushRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 109 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 1 | BOOLEAN | flush |
| 3 | | unused |

### 10.3.17 Free Renderer

5.0

**Name**

**PEXFreeRenderer**

**Request**

*rdr_id*: RENDERER_ID

**Errors**

Renderer
*rdr_id* is invalid.

**Description**

This request deletes the specified renderer resource and frees the storage associated with it. If the renderer’s *renderer state* is *Rendering* or *MultipassRendering*, an implicit **PEXEndRendering** request with *flush* equal to *False* is performed before the renderer is freed. If the renderer’s *renderer state* is *Picking*, the pick operation is aborted.

**Encoding: PEXFreeRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 20 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |

### 10.3.18 Get Renderer Attributes

5.0

**Name**

**PEXGetRendererAttributes**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *rdr_id*: | RENDERER_ID |
| *item_mask*: | BITMASK |

**Reply**

| | |
|---|---|
| *item_list*: | LISTofVALUE |

**Errors**

Renderer
*rdr_id* is invalid.

Value
An invalid bit is set in *item_mask*.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

**Description**

This request returns the renderer attributes indicated by *item_mask* (§10.1.5) for the renderer identified by *rdr_id*. The attributes are returned in *item_list*. Floating point values in *item_list* are returned in the floating point format specified by *fp_format*.

If a pipeline context, lookup table, or name set resource ID is requested by this request, and that resource has been freed, the value *AlreadyFreed* (§3.7.7.120) is returned as the ID of that resource.

**Encoding: PEXGetRendererAttributes**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 22 | PEX opcode |
| 2 | 4+*s*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 4 | BITMASK | item mask |
| *s*†1 | BITMASK | item mask second word 5.2 |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *n*/4 | reply length |
| 24 | | unused |
| *n* | LISTofVALUE | item list |

†1) *s* = ((0x80000000 & item mask) ? 4 : 0)

### 10.3.19 Get Z-Buffer

**Name**

**PEXGetZBuffer**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *rdr_id*: | RENDERER_ID |
| *x, y*: | INT16 |
| *width*, *height*: | CARD16 |

**Reply**

| | |
|---|---|
| *values*: | LISTofVALUE |

**Errors**

Renderer
*rdr_id* is invalid.

Match
The drawable associated with the renderer cannot have a Z-buffer.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Alloc
The server was unable to allocate a Z-buffer.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is *Rendering* or *MultipassRendering*, the content of the given rectangle of the Z-buffer is returned. The validity of values returned if the *renderer state* is *Picking* is implementation-dependent. If the *renderer state* is *Idle*, a zero-length list of *values* is returned.

The *x* and *y* coordinates, given in DCs, are relative to the renderer's current ***drawable origin*** and define the ***upper-left*** corner of the rectangle. The *width* and *height*, given in pixels, are inclusive. It is undefined what is copied for obscured regions of the drawable associated with the renderer. If the rectangle extends beyond the drawable, a zero-length list of *values* is returned. The renderer's *clip list* is not considered by this request.

The values are returned row by row, starting with the upper left of the rectangle. The returned Z-buffer values are mapped from device specific values into the range of [0.0, 1.0] in DC space (§2.2.1.4). The returned *values* are of type FLOAT, in the floating-point format specified in *fp_format*.

**Encoding: PEXGetZBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 114 | PEX opcode |
| 2 | 5 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 2 | INT16 | x |
| 2 | INT16 | y |
| 2 | CARD16 | width |
| 2 | CARD16 | height |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *n*fp*/4 | reply length |
| 4 | CARD32 | number of values (*n*) |
| 20 | | unused |
| *n*fp* | LISTofVALUE | values |

### 10.3.20 Init Multipass

**Name**

**PEXInitMultipass**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |

**Errors**

Renderer
*rdr_id* is invalid.

RendererState
The renderer state is already *MultipassRendering*.

**Description**

If the *renderer state* of the renderer identified by *rdr_id* is *Rendering*, this request changes *renderer state* to *MultipassRendering*. The renderer is initialized to perform multipass rendering. The first pass is defined as all requests received by the renderer subsequent to **PEXInitMultipass** and prior to **PEXNextPass** or **PEXNextPassWithoutReply**. Subsequent passes are defined as the requests between sequential invocations of **PEXNextPass** or **PEXNextPassWithoutReply**.

If *renderer state* is already *MultipassRendering* when this request is invoked, the request performs the operations listed above and a *RendererState* error is generated.

Support for multipass rendering is implementation-dependent, and can be determined by corresponding support of capabilities that require multipass rendering, such as the transparency method *TransparencyMultipass*.

If the *renderer state* is *Idle* or *Picking* or if multipass rendering is not supported, this request has no effect.

**Encoding: PEXInitMultipass**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 111 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |

### 10.3.21 Init Renderer

**Name**

**PEXInitRenderer**

**Request**

*rdr_id*: RENDERER_ID

**Errors**

Renderer
*rdr_id* is invalid.

**Description**

This request sets the pipeline state of the renderer identified by *rdr_id* to the values in the pipeline context associated with the renderer. Any saved pipeline state values (e.g. from a **PEXBeginStructure** request) are lost. If no pipeline context is associated with the renderer, the default pipeline context values listed in §9.1 are used. Also, the renderer's *current path* is initialized to a single element reference with a structure identifier of zero and an offset of zero.

If the renderer's *renderer state* is *Idle*, this request has no effect.

**Encoding: PEXInitRenderer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 110 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |

### 10.3.22 Next Pass

**Name**

**PEXNextPass**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *next_pass*: | {*MultipassContinue*, *MultipassLastPass*, *MultipassAbort*} |

**Reply**

| | |
|---|---|
| *count*: | INT32 |

**Errors**

Renderer
*rdr_id* is invalid.

Value
*next_pass* contains an invalid value.

**Description**

This request signifies the end of a multipass-rendering pass for the renderer identified by *rdr_id*, and, if applicable, the beginning of the next pass for that renderer. The renderer executes any processing necessary to finish the pass. The number of additional passes required, if known, is returned in *count*.

If *next_pass* is *MultipassContinue*, the server prepares to receive the next pass, which the client is expected to send unless *count* is 0.

If *next_pass* is *MultipassLastPass*, the server prepares to complete the multipass operation with only one more pass. The server produces the best output it can using the multipass algorithms it is currently using for the multipass session. The client is expected to send the additional pass unless *count* is 0.

If *next_pass* is *MultipassAbort*, the server terminates the multipass session. The effect on the rendered image is implementation-dependent. The *renderer state* is changed to *Rendering*. The number of additional passes that were required, if known, for this multipass session are returned in *count*.

If the client sends an additional pass when that pass was not required, the results are implementation-dependent.

Except where defined otherwise in this description, the returned *count* indicates whether additional passes are needed to complete the current multipass session. A value of zero indicates that rendering is complete and that this request caused the renderer state to become *Rendering*. A value of −1 indicates that the number of passes needed to complete rendering is unknown, and the client is expected to send passes until a count of zero is returned.

When a multipass session ends, whether fully completed or prematurely terminated, all resources other than the frame buffer, such as the Z-buffer and accumulation buffers, are left in an implementation-dependent state. Therefore, continuing in single-pass rendering mode after completing multipass will typically require initialization of all applicable state and resources.

If *renderer state* is not *MultipassRendering* when this request is invoked, it remains unchanged and zero is returned in *count*. No other action is taken.

**Encoding: PEXNextPass**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 112 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 2 | CARD16 | multipass control |
| | 0 *MultipassAbort* | |
| | 1 *MultipassLastPass* | |
| | 2 *MultipassContinue* | |
| 2 | | unused |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 0 | reply length |
| 4 | INT32 | count |
| 20 | | unused |

### 10.3.23 Next Pass Without Reply

**Name**

**PEXNextPassWithoutReply**

**Request**

*rdr_id*: RENDERER_ID

*next_pass*: {*MultipassContinue*, *MultipassLastPass*, *MultipassAbort*}

**Errors**

Renderer
*rdr_id* is invalid.

Value
*next_pass* contains an invalid value.

**Description**

This request behaves the same as **PEXNextPass** with the single exception being that no reply is sent back to the client.

**Encoding: PEXNextPassWithoutReply**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 113 | PEX opcode |
| 2 | 3 | request length |
| 4 | RENDERER_ID | renderer ID |
| 2 | CARD16 | multipass control |
| | 0 *MultipassAbort* | |
| | 1 *MultipassLastPass* | |
| | 2 *MultipassContinue* | |
| 2 | | unused |

### 10.3.24 Put Z-Buffer

**Name**

**PEXPutZBuffer**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *rdr_id*: | RENDERER_ID |
| *x, y*: | INT16 |
| *width*, *height*: | CARD16 |
| *values*: | LISTofVALUE |

**Errors**

Renderer
*rdr_id* is invalid.

Match
The drawable associated with the renderer cannot have a Z-buffer.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Alloc
The server was unable to allocate a Z-buffer.

**Description**

If the *renderer state* attribute of the renderer indicated by *rdr_id* is *Rendering* or *MultipassRendering*, the described rectangle of the Z-buffer is written with the *values* given. If the *renderer state* is *Picking*, the *values* written will have an implementation-dependent effect. If the *renderer state* is *Idle*, nothing is written to the Z-buffer.

The *x* and *y* coordinates, given in DCs, are relative to the renderer’s current ***drawable origin*** and define the ***upper-left*** corner of the rectangle. The *width* and *height*, given in pixels, are inclusive. Nothing is written for regions outside or obscured regions of the drawable. The renderer’s *clip list* is not considered by this request.

The *values* are ordered row by row, starting at the upper left of the rectangle. The supplied Z-buffer values are specified in the range of [0.0, 1.0] in DC space (§2.2.1.4) and are mapped into device-specific values when written. The *values* are of type FLOAT, in the floating-point format specified in *fp_format*.

**Encoding: PEXPutZBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 115 | PEX opcode |
| 2 | 6 + *n*fp*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 2 | INT16 | x |
| 2 | INT16 | y |
| 2 | CARD16 | width |
| 2 | CARD16 | height |
| 4 | CARD32 | number of values (*n*) |
| *n*fp* | LISTofVALUE | Z-buffer values |

### 10.3.25 Render Elements 5.1

**Name**

**PEXRenderElements**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *s_id*: | STRUCTURE_ID |
| *range*: | ELEMENT_RANGE |

**Errors**

Renderer
*rdr_id* is invalid.

Structure
*s_id* is invalid.

StructurePermission
An invalid position in a locked structure (§ 12.1.1.3) is given in *range*; *whence* values must be *Beginning* and *offset* values must be 0 or select structure references.

Value
A whence value in *range* is invalid.

**Description**

If the *renderer state* of the renderer specified by *rdr_id* is set to *Rendering*, *Picking* or *MultipassRendering*, the renderer traverses and renders the elements indicated by *range* in the structure specified by *s_id*. If the *renderer state* is *Idle*, this request has no effect. The semantics of this request are the same as for **PEXRenderOutputCommands** except that the output commands are taken from the specified structure.

If position 0 is given as the start of the element range, the implicit element at position 0 is ignored and rendering commences with the first element.

**Encoding: PEXRenderElements**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 97 | PEX opcode |
| 2 | 7 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | STRUCTURE_ID | structure ID |
| 16 | ELEMENT_RANGE | element range (§ 12.2.1) |

### 10.3.26 Render Network

**Name**

**PEXRenderNetwork**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |
| *s_id*: | STRUCTURE_ID |

**Errors**

Renderer
*rdr_id* is invalid.

Drawable
*drawable_id* is invalid.

Structure
*s_id* is invalid.

Match
1) The specified renderer was not created for drawables with the root, depth, and visual ID of *drawable_id*.
2) The specified drawable is of an unsupported type.

Alloc
The server is unable to allocate the necessary resources.

RendererState
The renderer's renderer state is *Rendering*, *Picking* or *MultipassRendering*.

**Description**

This request causes the structure network rooted at *s_id* to be traversed and rendered on *drawable_id*, using the renderer specified by *rdr_id*. The request is equivalent to invoking the following sequence of requests, except the structure identifier of the first element of the *current path* renderer attribute is initialized to *s_id* instead of to zero:

- **PEXBeginRendering**(*rdr_id*, *drawable_id*)
- **PEXRenderElements**(*rdr_id*, *s_id*, all elements of *s_id*)
- **PEXEndRendering**(*rdr_id*, *True*)

If the renderer state is not *Idle*, the *RendererState* error is issued, the operation which was executing is aborted, and the structure network rooted as *s_id* is traversed and rendered as described above.

**Encoding: PEXRenderNetwork**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 29 | PEX opcode |
| 2 | 4 | request length |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |
| 4 | STRUCTURE_ID | structure ID |

### 10.3.27 Render Output Commands

5.0

**Name**

**PEXRenderOutputCommands**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *rdr_id*: | RENDERER_ID |
| *cmds*: | LISTofOUTPUT_CMD |

**Errors**

Renderer
*rdr_id* is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

OutputCommand
One or more of the output commands contains an invalid value.

**Description**

If the *renderer state* of the renderer identified by *rdr_id* is *Rendering*, *Picking* or *MultipassRendering*, the output commands in *cmds* are immediately processed. If the *renderer state* is *Idle*, this request has no effect.

Floating-point values in *cmds* are in the floating-point format specified in *fp_format*.

Output commands are processed by the server in order until the list is exhausted or until an output command is found to be in error. In the case of an output command error, the erroneous output command and all output commands subsequent to it in *cmds* are ignored and an *OutputCommand* error is generated. The *current path* attribute of the renderer is updated to reflect only those output commands processed before an error.

**Encoding: PEXRenderOutputCommands**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 28 | PEX opcode |
| 2 | 4+*s*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 4 | CARD32 | number of output commands (*n*) |
| *s* | LISTofOUTPUT_CMD | cmds |

## 10.4 Requests for Versions 5.0 and 5.1 Compatibility

The **PEXGetRendererDynamics** request is deprecated in version 5.2 because PEX 5.2 requires that implementations make all renderer attributes dynamic. However, the request is maintained for compatibility with 5.0 and 5.1 clients.

### 10.4.1 Get Renderer Dynamics

5.0

**Name**

**PEXGetRendererDynamics**

**Request**

*rdr_id*: RENDERER_ID

**Reply**

*tables*: BITMASK

*name_sets*: BITMASK

*attributes*: BITMASK

**Errors**

Renderer
*rdr_id* is invalid.

**Description**

This request returns the dynamics (binding times) of all renderer attributes specified prior to version 5.2 for the renderer identified by *rdr_id*. The dynamics are returned in three bitmasks: *tables*, *name_sets*, and *attributes*. For each defined bit in these masks, a value of 0 indicates that the corresponding attribute is dynamic, a value of 1 indicates that it is not.

Changes to dynamic attributes take effect immediately. Attributes that are not dynamic may be modified at any time but the new value does not take effect until the next explicit or implicit **PEXBeginRendering**. Implementations of PEX 5.2 are required to make all renderer tables, name sets and attributes dynamic. The semantics of switching between supported values of *HLHSR mode* are implementation-dependent.

**Encoding: PEXGetRendererDynamics**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 23 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 0 | reply length |

**Encoding: PEXGetRendererDynamics (Continued)**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 4 | BITMASK | | tables |
| | 0x00000001 | *DynMarkerBundle* | |
| | 0x00000002 | *DynTextBundle* | |
| | 0x00000004 | *DynLineBundle* | |
| | 0x00000008 | *DynInteriorBundle* | |
| | 0x00000010 | *DynEdgeBundle* | |
| | 0x00000020 | *DynViewTable* | |
| | 0x00000040 | *DynColorTable* | |
| | 0x00000080 | *DynDepthCueTable* | |
| | 0x00000100 | *DynLightTable* | |
| | 0x00000200 | *DynColorApproxTable* | |
| | 0x00000400 | *DynPatternTable* | |
| | 0x00000800 | *DynTextFontTable* | |
| | 0x00010000 | *DynMarkerBundleContents* | |
| | 0x00020000 | *DynTextBundleContents* | |
| | 0x00040000 | *DynLineBundleContents* | |
| | 0x00080000 | *DynInteriorBundleContents* | |
| | 0x00100000 | *DynEdgeBundleContents* | |
| | 0x00200000 | *DynViewTableContents* | |
| | 0x00400000 | *DynColorTableContents* | |
| | 0x00800000 | *DynDepthCueTableContents* | |
| | 0x01000000 | *DynLightTableContents* | |
| | 0x02000000 | *DynColorApproxTableContents* | |
| | 0x04000000 | *DynPatternTableContents* | |
| | 0x08000000 | *DynTextFontTableContents* | |
| 4 | BITMASK | | name sets |
| | 0x00000001 | *DynHighlightNameset* | |
| | 0x00000002 | *DynInvisibilityNameset* | |
| | 0x00000004 | *DynPickNameset* | |
| | 0x00010000 | *DynHighlightNamesetContents* | |
| | 0x00020000 | *DynInvisibilityNamesetContents* | |
| | 0x00040000 | *DynPickNamesetContents* | |
| 4 | BITMASK | | attributes |
| | 0x00000001 | *DynHLHSRMode* | |
| | 0x00000002 | *DynNPCSubvolume* | |
| | 0x00000004 | *DynViewport* | |
| | 0x00000008 | *DynClipList* | |
| | 0x00000010 | *DynEchoMode* | |
| 12 | | | unused |

# 11 Renderer Picking

## 11.1 General

Picking is similar to rendering, with the exception that primitives are not rendered during picking but are instead subjected to pick criteria that determine whether they are “picked.” The identity of primitives that match the criteria are returned to the client.

During picking, the renderer’s *renderer state* is set to *Picking*. No primitives are drawn when the renderer is in this state.

### 11.1.1 Detectable Primitives

A primitive passed to a renderer during picking is detectable at a point on the primitive if:

a) The primitive is not invisible, as defined by the renderer’s invisibility inclusion and exclusion name sets.

b) The primitive has one of its names in the renderer’s pick inclusion name set and none of its names in the pick exclusion set.

c) The point on the primitive is not part of a culled face.

d) The point on the primitive is not clipped by the model clipping volume, view clipping planes, the NPC limits, the renderer’s clip list, or the drawable’s boundaries.

Picking uses the geometric definition of primitives, so area primitives with interior style *Hollow* or *Empty* are detectable. Transparent primitives are also detectable. It is implementation-dependent whether rendering effects, such as linewidth, that increase the area of the primitive are considered during picking.

### 11.1.2 Pick Criteria

Detectable primitives are picked if they also meet the criteria of a pick device, specified when the pick is initiated. Two types of pick devices are defined (Table 11-3 ). The supported types are returned by the **PEXGetEnumeratedTypeInfo** request.

**Table 11-1 . Enumerated Type: *PickDeviceType***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *DCHitBox* 5.1 | See §11.1.2.1. |
| 2 | *NPCHitVolume* 5.1 | See §11.1.2.2. |

#### 11.1.2.1 DCHitBox Pick Device

Primitives are picked by a *DCHitBox* device if any detectable portion of the primitive intersects an aperture in device coordinates (§2.2.1.4). The aperture is specified as a 2D location and a distance, which defines a cylinder or parallelepiped extending from Z=0 to Z=1 with its center through the 2D location and a radius or half-width equal to the given distance. Whether the aperture is a cylinder or parallelepiped is implementation-dependent.

#### 11.1.2.2 NPCHitVolume Pick Device

Primitives are picked by an *NPCHitVolume* device if any detectable portion intersects an aperture in normalized projection coordinates (NPC). The aperture is a parallelpiped defined by two points giving its opposite corners.

### 11.1.3 Pick All and Pick One

Requests are defined to pick one primitive matching the pick criteria or all primitives matching the criteria. These requests are listed in Table 11-2 . The **PEXBeginPick**- and **PEXEndPick**- are counterparts of the renderer requests **PEXBeginRendering** and **PEXEndRendering**. The **PEXPick**- requests are counterparts of **PEXRenderNetwork**.

**Table 11-2 . Rendering Picking Requests**

| Request | Description |
|---|---|
| **PEXBeginPickOne** | Set the renderer state to *Picking* and prepare to pick one primitive that matches the pick criteria. |
| **PEXEndPickOne** | Terminate the pick begun with **PEXBeginPickOne** and return the picked primitive. |
| **PEXPickOne** | Pick one primitive from those in a structure network that match the pick criteria. |
| **PEXBeginPickAll** | Set the renderer state to *Picking* and prepare to pick all primitives that match the pick criteria. |
| **PEXEndPickAll** | Terminate the pick begun with **PEXBeginPickAll** and return the picked primitives. |
| **PEXPickAll** | Pick all primitives in a structure network that match the pick criteria. |

#### 11.1.3.1 Pick One

The identity of one primitive matching the pick criteria is returned for a pick-one pick. Which primitive to return of those matching the criteria is governed by the pick method. The defined pick-one pick methods are listed in Table 11-3 . The supported pick-one methods are returned by **PEXGetEnumeratedTypeInfo**. It is implementation-dependent whether the contents of a Z-buffer associated with the drawable passed to the **PEXBeginPickOne** or **PEXPickOne** requests are preserved during a pick using one of the *Visible*- pick methods.

**Table 11-3 . Enumerated Type: *PickOneMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *Last* | The last picked primitive is returned. 5.1 |
| 2 | *ClosestZ* | The primitive which has the Z value closest to the front clipping plane in the pick aperture is returned. If several primitives are equally close to the front clipping plane then it’s implementation-dependent which one of those primitives is returned. 5.1 |
| 3 | *VisibleAny* | A picked primitive that is visible after performing HLHSR is returned. If more than one is visible then it is implementation-dependent which one of those is returned. 5.1 |
| 4 | *VisibleClosest* | The primitive which is both visible after performing HLHSR and closest to the pick position is returned. The algorithm for determining which primitive is closest to the pick position is implementation-dependent. (For *NPCHitVolume* pick devices, the pick position is the center of the aperture.) 5.1 |

#### 11.1.3.2 Pick All

The identity of all picked primitives, up to a limit specified in the initializing request, are returned for a pick-all pick. A pick method that further qualifies which primitives are returned is specified when initiating the pick. The pick-all pick methods are listed in Table 11-4 . The supported pick methods are returned by **PEXGetEnumeratedTypeInfo**. It is implementation-dependent whether the contents of a Z-buffer associated with the drawable passed to **PEXBeginPickAll** or **PEXPickAll** request are preserved during a pick using the *Visible* pick method.

**Table 11-4 . Enumerated Type: *PickAllMethod***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *All* | All primitives that intersect the pick aperture are returned. 5.1 |
| 2 | *Visible* | Only primitives that intersect the pick aperture and are visible after HLHSR is performed are returned. 5.1 |

### 11.1.4 Response to Drawable Changes

If the drawable passed to a picking request is destroyed or resized while an immediate mode pick is being conducted, picking is terminated and the renderer state is set to *Idle*. All subsequent output and traversal commands are ignored until the appropriate **PEXEndPickOne** or **PEXEndPickAll** request is received. No errors are generated, but the returned pick status is *Aborted*.

If the drawable associated with a renderer is moved, exposed, or occluded while the renderer state is *Picking*, the server processes subsequent output commands using the newly-modified drawable.

## 11.2 Renderer Picking Requests

The requests that initiate and terminate renderer picking are defined in this section.

### 11.2.1 Begin Pick One

5.1

**Name**

**PEXBeginPickOne**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pick_method*: | PICK_ONE_METHOD |
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |
| *s_id*: | INT32 |
| *pick_data*: | PICK_RECORD |

**Errors**

Alloc
The server was unable to allocate the resources necessary to do picking.

Renderer
*rdr_id* is invalid.

Drawable
*drawable_id* is invalid.

Match
1) The specified renderer was not created for the same root, depth, and visual ID as the specified drawable.
2) The specified drawable is of an unsupported type.

RendererState
The renderer’s renderer state is not *Idle*.

Value
The *pick_method* or *pick_data* contains an invalid value.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

**Description**

This request initializes the renderer identified by *rdr_id* for pick-one picking and sets its renderer state to *Picking*. Output commands subsequently passed to the renderer are processed as if the renderer state were *Rendering*, with the exception that output primitive output commands are not rendered but instead examined to determine whether they are detectable and match the pick criteria. *pick_method* specifies the method used to select one primitive from those matching the pick criteria (§ 11.1.3.1). *pick_data* indicates the pick device type (§ 11.1.2) to use.

The renderer’s *current path* is initialized to one entry with a structure member of *s_id* and an offset of 0. The value in *s_id* need not identify a PEX structure; it is a value chosen by the client. The pick path returned from a pick-one traversal is equivalent to the renderer’s *current path* at the time the picked primitive was processed.

Floating-point values in *pick_data* are specified in the floating-point format indicated by *fp_format*.

If the renderer's *renderer state* is *Rendering*, *Picking* or *MultipassRendering*, a *RendererState* error is generated, but the pick is also initiated as defined above.

**Encoding: PEXBeginPickOne**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 99 | PEX opcode |
| 2 | 6+*n* | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | PICK_ONE_METHOD | pick-one method |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |
| 4 | INT32 | structure identifier |
| 4+4*n* | PICK_RECORD | pick data (§3.7.7.105) |

### 11.2.2 End Pick One

5.1

**Name**

**PEXEndPickOne**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |

**Reply**

| | |
|---|---|
| *pick_status*: | {*NoPick*, *Ok*, *Aborted*} |
| *better_pick*: | BOOLEAN |
| *pick_path*: | LISTofPICK_ELEMENT_REF |

**Errors**

Renderer
*rdr_id* is invalid.

RendererState
The renderer state is *Rendering*, *MultipassRendering* or was set to *Picking* by a **PEXBeginPickAll** request.

**Description**

This request terminates the pick-one state initiated by a **PEXBeginPickOne** request for the renderer identified by *rdr_id*. The renderer's *renderer state* is changed to *Idle*. If a primitive was picked, the *pick_status* returned is *Ok* and *pick_path* contains the value of the renderer's *current path* at the time the picked primitive was detected by the renderer. If no primitive was picked then *pick_status* is *NoPick* and *pick_path* is empty. If the renderer's drawable has been destroyed or resized since the initiating **PEXBeginPickOne** request, the *pick_status* returned is *Aborted* and *pick_path* is empty.

If a primitive more closely satisfied the pick criteria than the one returned but was designated undetectable by the renderer's pick inclusion and exclusion name sets, *better_pick* is returned as *True*. Otherwise *better_pick* is *False*.

If the renderer's *renderer state* is *Rendering*, *MultipassRendering* or pick-all is in progress, a *RendererState* error is generated, but this request otherwise has no effect. If *renderer state* is *Idle* when this request is received, the request is ignored and no error is generated.

**Encoding: PEXEndPickOne**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 100 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 3*$n$ | reply length |
| 4 | CARD32 | number of entries in pick path ($n$) |
| 2 | CARD16 | pick status |
| | 0 *NoPick* 5.1 | |
| | 1 *Ok* 5.1 | |
| | 2 *Aborted* 5.1 | |
| 1 | BOOLEAN | better pick |
| 17 | | unused |
| 12*$n$ | LISTofPICK_ELEMENT_REF | pick path (§3.7.7.103) |

### 11.2.3 Pick One

5.1

**Name**

**PEXPickOne**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pick_method*: | PICK_ONE_METHOD |
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |
| *s_id*: | STRUCTURE_ID |
| *pick_data*: | PICK_RECORD |

**Reply**

| | |
|---|---|
| *pick_status*: | {*NoPick*, *Ok*, *Aborted*} |
| *better_pick*: | BOOLEAN |
| *pick_path*: | LISTofPICK_ELEMENT_REF |

**Errors**

Alloc
The server was unable to allocate the resources necessary to do picking.

Renderer
*rdr_id* is invalid.

Structure
s_id is invalid.

Drawable
*drawable_id* is invalid.

Match
1) The specified renderer was not created for the same root, depth, and visual ID as the specified drawable.
2) The specified drawable is of an unsupported type.

RendererState
The renderer's renderer state is not *Idle*.

Value
The *pick_method* or *pick_data* contains an invalid value.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

### Description

This request performs a pick traversal of the structure network rooted at *s_id* and returns at most one picked primitive. It is equivalent to the following sequence of requests:

- **PEXBeginPickOne**(*pick_method*, *rdr_id*, *drawable_id*, *s_id*, *pick_data*)
- **PEXRenderElements**(*rdr_id*, *s_id*, all elements of *s_id*)
- **PEXEndPickOne**(*rdr_id*)

### Encoding: PEXPickOne

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 101 | PEX opcode |
| 2 | 6+*n* | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | PICK_ONE_METHOD | pick-one method |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |
| 4 | STRUCTURE_ID | structure ID |
| 4+4*n* | PICK_RECORD | pick data (§3.7.7.105) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 3*n* | reply length |
| 4 | CARD32 | number of entries in pick path (*n*) |
| 2 | CARD16 | pick status |
| | 0 *NoPick* 5.1 | |
| | 1 *Ok* 5.1 | |
| | 2 *Aborted* 5.1 | |
| 1 | BOOLEAN | better pick |
| 17 | | unused |
| 12*n* | LISTofPICK_ELEMENT_REF | pick path (§3.7.7.103) |

### 11.2.4 Begin Pick All

5.1

**Name**

**PEXBeginPickAll**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pick_method*: | PICK_ALL_METHOD |
| *send_event*: | BOOLEAN |
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |
| *s_id*: | INT32 |
| *pick_max_hits*: | CARD32 |
| *pick_data*: | PICK_RECORD |

**Errors**

Alloc
The server was unable to allocate the resources necessary to do picking.

Renderer
*rdr_id* is invalid.

Drawable
*drawable_id* is invalid.

Match
1) The specified renderer was not created for the same root, depth, and visual ID as the specified drawable.
2) The specified drawable is of an unsupported type.

RendererState
The renderer’s renderer state is not *Idle*.

Value
The *pick_method* or *pick_data* contains an invalid value.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

**Description**

This request initializes the renderer identified by *rdr_id* for pick-all picking and sets its renderer state to *Picking*. Output commands subsequently passed to the renderer are processed as if the renderer state were *Rendering*, with the exception that output primitive output commands are not rendered but instead examined to determine whether they are detectable and match the pick criteria. *pick_method* specifies the method used to select primitives from those matching the pick criteria (§ 11.1.3.1). *pick_data* indicates the pick device type (§ 11.1.2) to use.

The renderer’s *current path* is initialized to one entry with a structure member of *s_id* and an offset of 0. The value in *s_id* need not identify a PEX structure; it is a value chosen by the client. The pick paths returned from a pick-all traversal are equivalent to the renderer’s *current path* at the time each picked primitive was processed.

The identities of the first *pick_max_hits* primitives matching the pick criteria are recorded. If *send_event* is *True* and *pick_method* is *All* (as opposed to *Visible)*, a ***MaxHitsReached*** event is generated when the number of primitives indicated by *pick_max_hits* are picked, if that event is supported by the server. **PEXGetImpDepConstants** indicates whether the event is supported as well as the maximum total number of pick element references supported (**MaxNumPickResults**). Upon receiving the event, the client should end the traversal and see if the recorded selections are sufficient.

Floating-point values in *pick_data* are specified in the floating-point format indicated by *fp_format*.

If the renderer's *renderer state* is *Rendering*, *Picking* or *MultipassRendering*, a *RendererState* error is generated, but the pick is also initiated as defined above.

**Encoding: PEXBeginPickAll**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 102 | PEX opcode |
| 2 | 8+*n* | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | PICK_ALL_METHOD | pick-all method |
| 1 | BOOLEAN | send event |
| 3 | | unused |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |
| 4 | INT32 | structure identifier |
| 4 | CARD32 | pick max hits |
| 4+4**n* | PICK_RECORD | pick data (§3.7.7.105) |

### 11.2.5 End Pick All

5.1

**Name**

**PEXEndPickAll**

**Request**

| | |
|---|---|
| *rdr_id*: | RENDERER_ID |

**Reply**

| | |
|---|---|
| *pick_status*: | {*NoPick*, *Ok*, *Aborted*} |
| *more_picks*: | {*MoreHits*, *NoMoreHits*, *MayBeMoreHits*} |
| *pick_paths*: | LISTofLISTofPICK_ELEMENT_REF |

**Errors**

Renderer
*rdr_id* is invalid.

RendererState
The renderer state is *Rendering*, *MultipassRendering* or was set to *Picking* by a **PEXBeginPickOne** request.

**Description**

This request terminates the pick-all state initiated by a **PEXBeginPickAll** request for the renderer identified by *rdr_id*. The renderer's *renderer state* is changed to *Idle*. If one or more primitives were picked, the *pick_status* returned is *Ok* and *pick_paths* contains the values of the renderer's *current path* at the time each picked primitive was detected by the renderer. However if a picked primitive is in a locked structure (§ 12.1.1.3), the primitive's element position is undefined. If no primitive was picked, *pick_status* is *NoPick* and *pick_paths* is empty. If the renderer's drawable has been destroyed or resized since the initiating **PEXBeginPickAll** request, the *pick_status* returned is *Aborted* and *pick_paths* is empty.

If all picks were recorded than *more_picks* is set to *NoMoreHits* and the renderer's *pick start path* attribute is made empty. If the maximum number of hits was reached, or the maximum number of pick path elements used, and additional picks were detected, the value *MoreHits* is returned in *more_picks*. If the server does not know whether there were detections afterwards, the value *MayBeMoreHits* is returned in *more_picks*. If *more_picks* is set to either *MoreHits* or *MayBeMoreHits*, the renderer sets its *pick start path* to allow the client to invoke **PEXBeginPickAll** again to search for additional primitives, starting its traversal with the element following the element identified by the *pick start path*. Unless the last primitive picked is in a locked structure, the renderer sets the *pick start path* to the last primitive picked. If the primitive's structure is locked, the *pick start path* is set to the element preceding the first structure reference in the locked structure after the picked primitive, or to the element referencing the locked structure if there are no later structure references in the locked structure.

If the renderer's *renderer state* is *Rendering*, *MultipassRendering* or a pick-one is in progress, a *RendererState* error is generated, but this request otherwise has no effect. If *renderer state* is *Idle* when this request is received, the request is ignored and no error is generated.

**Encoding: PEXEndPickAll**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 103 | PEX opcode |
| 2 | 2 | request length |
| 4 | RENDERER_ID | renderer ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $n+3*m$[†1] | reply length |
| 4 | CARD32 | number of pick paths ($n$) |
| 2 | CARD16 | pick status |
| | 0 *NoPick* 5.1 | |
| | 1 *Ok* 5.1 | |
| | 2 *Aborted* 5.1 | |
| 2 | CARD16 | more picks |
| | 0 *MoreHits* 5.1 | |
| | 1 *NoMoreHits* 5.1 | |
| | 2 *MayBeMoreHits* 5.1 | |
| 16 | | unused |
| $4*n+12*m$ | LISTofLISTofPICK_ELEMENT_REF | pick paths |

†1) $m$ is the number of pick element references.

### 11.2.6 Pick All 5.1

**Name**

**PEXPickAll**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *pick_method*: | PICK_ALL_METHOD |
| *rdr_id*: | RENDERER_ID |
| *drawable_id*: | DRAWABLE_ID |
| *pick_max_hits*: | CARD32 |
| *pick_data* | PICK_RECORD |

**Reply**

| | |
|---|---|
| *pick_status*: | {*NoPick*, *Ok*, *Aborted*} |
| *more_picks*: | {*MoreHits*, *NoMoreHits*, *MayBeMoreHits*} |
| *pick_paths*: | LISTofLISTofPICK_ELEMENT_REF |

**Errors**

Alloc
The server was unable to allocate the resources necessary to do picking.

Renderer
*rdr_id* is invalid.

Drawable
*drawable_id* is invalid.

Match
1) The specified renderer was not created for the same root, depth, and visual ID as the specified drawable.
2) The specified drawable is of an unsupported type.

RendererState
The renderer’s renderer state is not *Idle*.

Value
The *pick_method* or *pick_data* contains an invalid value.

Path
The renderer’s *pick start path* is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

StructurePermission
The renderer’s *pick start path* selects an invalid position in a locked structure.

**Description**

This request traverses the structure network specified in the renderer's *pick start path* attribute, beginning with the element after the one at the end of that path and proceeding to the last element of the first structure identified in *pick start path*. The renderer's *pick start path* can be initialized with requests **PEXCreateRenderer** and **PEXChangeRenderer**.

If one or more primitives are picked, the *pick_status* returned is *Ok* and *pick_paths* contains the path through the network to the picked primitive for each primitive picked. However if a picked primitive is in a locked structure (§ 12.1.1.3), the primitive's element position is undefined. If no primitive was picked, *pick_status* is *NoPick* and *pick_paths* is empty. If the renderer's drawable has been destroyed or resized since the initiating **PEXBeginPickAll** request, the *pick_status* returned is *Aborted* and *pick_paths* is empty.

If all picks were recorded than *more_picks* is set to *NoMoreHits* and the renderer's *pick start path* attribute is made empty. If the maximum number of hits was reached, or the maximum number of pick path elements used, and additional picks were detected, the value *MoreHits* is returned in *more_picks*. If the server does not know whether there were detections afterwards, the value *MayBeMoreHits* is returned in *more_picks*. If *more_picks* is set to either *MoreHits* or *MayBeMoreHits*, the renderer sets its *pick start path* to allow the client to invoke **PEXPickAll** again to search for additional primitives. Unless the last primitive picked is in a locked structure (§ 12.1.1.3), the renderer sets the *pick start path* to the last primitive picked. If the primitive's structure is locked, the *pick start path* is set to the element preceding the first structure reference in the structure after the picked primitive, or to the element referencing the locked structure if there are no later structure references in the locked structure.

**PEXGetImpDepConstants** indicates the maximum total number of pick element references supported (**MaxNumPickResults**).

**Encoding: PEXPickAll**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 104 | PEX opcode |
| 2 | 6+$n$ | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | PICK_ALL_METHOD | pick-all method |
| 4 | RENDERER_ID | renderer ID |
| 4 | DRAWABLE_ID | drawable ID |
| 4 | CARD32 | pick max hits |
| 4+4*$n$ | PICK_RECORD | pick data |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $n$+3*$m$[†1] | reply length |
| 4 | CARD32 | number of pick paths ($n$) |
| 2 | CARD16 | pick status |
| | 0 *NoPick* 5.1 | |
| | 1 *Ok* 5.1 | |
| | 2 *Aborted* 5.1 | |
| 2 | CARD16 | more picks |
| | 0 *MoreHits* 5.1 | |
| | 1 *NoMoreHits* 5.1 | |
| | 2 *MayBeMoreHits* 5.1 | |
| 16 | | unused |
| 4*$n$+12*$m$ | LISTofLISTofPICK_ELEMENT_REF | pick paths |

†1) $m$ is the number of pick element references.

# 12 Structures

## 12.1 Structures

Structures are resources that hold output commands for later execution. The requests used to create, manage, manipulate, and inquire structures are listed in Table 12-1 . The output commands are placed in a structure by the PEX client. Each output command within a structure is called a structure element. Structures are rendered when passed to a renderer in one of the following ways:

a) A **PEXRenderNetwork** request.

b) A **PEXRenderElements** request.

c) A **PEXRenderOutputCommands** request containing an Execute Structure, Conditional Execute Structure, Include Structure, or Conditional Include Structure output command.

Output commands are added to structures by the **PEXStoreElements** request (§ 12.2.22).

**Table 12-1 . Structure Resource Management Requests**

| Request | Description |
|---|---|
| **PEXCreateStructure** | Create a structure resource. |
| **PEXCopyElements** | Copy a range of elements in one structure to another structure. |
| **PEXDestroyStructures** | Delete structures and all references to them. |
| **PEXGetStructureInfo** | Retrieve a structure's attributes, size, and whether the structure is referenced. |
| **PEXGetElementInfo** | Retrieve the type and size of a range of elements of a structure. |
| **PEXGetStructuresInNetwork** | Retrieve either all or only the structures unique to the specified structure network. |
| **PEXGetAncestors** | Retrieve the unique ancestors of a structure, looking at the given range of paths. |
| **PEXGetDescendants** | Retrieve the unique descendants of a structure, looking at the given range of paths. |
| **PEXFetchElements** | Retrieve a range of elements from a structure. |
| **PEXSetEditingMode** | Set a structure's *editing mode* attribute. |
| **PEXSetElementPointer** | Set a structure's *element pointer* attribute. |
| **PEXSetElementPointerAtLabel** | Position a structure's element pointer at an offset from the next occurrence of a Label element. |
| **PEXSetElementPointerAtPickID** | Position a structure's element pointer at an offset from the next occurrence of a Pick Identifier element. |
| **PEXElementSearch** | Conduct a filtered search through a structure, from a specified position, for an element of a specified type. |
| **PEXSetStructurePermission** | Set a structure's *permission* attribute. |
| **PEXStoreElements** | Load elements into a structure. |
| **PEXDeleteElements** | Delete a range of elements from a structure. |

**Table 12-1 . Structure Resource Management Requests (Continued)**

| Request | Description |
|---|---|
| **PEXDeleteElementsToLabel** | Delete a range of elements, from the current position to the specified Label element, from a structure. |
| **PEXDeleteElementsBetweenLabels** | Delete a range of elements, specified by Label elements, from a structure. |
| **PEXCopyStructure** | Clear the destination structure, then copy all elements of the source structure into it. |
| **PEXChangeStructureReferences** | Modify all references to a structure to reference a different structure. |

### 12.1.1 Structure Attributes

Structures have three attributes that can be set by a client: *editing mode*, *element pointer*, and *permission*.

#### 12.1.1.1 Editing Mode

The editing mode indicates whether new elements are to replace existing elements in a structure or be inserted between them. A mode of *StructureInsert* causes elements to be inserted into a structure. An editing mode of *StructureReplace* causes new elements to replace existing elements. See §12.2.22 for the details. A structure’s editing mode defaults to *StructureInsert* when it is created. The mode is changed by a **PEXSetEditingMode** request.

#### 12.1.1.2 Element Pointer

The element pointer designates the location at which new elements are added to a structure. Each element within the structure is identified by its position. Position 1 is the first element in the structure. Elements beyond the first are assigned sequential positions. Position 0 corresponds to the location before the first element in the structure. An element’s position changes when elements between it and position 0 are added or removed. The element pointer can be positioned to any element in the structure or to position 0 with one of the following requests: **PEXElementSearch**, **PEXSetElement-Pointer**, **PEXSetElementPointerAtLabel**, or **PEXSetElementPointerAtPickID**. The element pointer is also changed when elements are added or removed. The requests that add and remove elements define their effect on the element pointer. A structure’s element pointer is initially 0.

#### 12.1.1.3 Permission

A structure’s *permission*, set by the client, places limits on what operations the client can conduct on that structure. The server can take advantage of these restrictions, typically to optimize the time it takes to render the structure. These optimizations are limited to the structure; the structure network is preserved.

By default a structure’s *permission* is *StructureReadWrite*. There are no limits on the operations which can be performed on a structure with permission *StructureReadWrite*.

When the client sets a structure’s *permission* to *StructureWriteOnly*, the ability is taken away for a client to read the structure’s elements or copy its elements to a structure whose permission is *StructureReadWrite*. Therefore an attempt to apply request **PEXFetchElement**, and, in some cases, requests **PEXCopyElements** and **PEXCopyStructure**, on a structure with permission *StructureWriteOnly* will results in a *StructurePermission* error. The server is free to modify an individual element as long as the structure network, the element’s type, and the element’s effect during rendering and picking is maintained.

When the client sets a structure’s *permission* to *StructureLocked*, the ability is taken away for a client to read, edit, and copy its elements. The server is free to modify the structure’s elements as long as the structure network and the structure’s effect during rendering and picking is maintained. For the purposes of rendering and picking, the only valid element positions of a locked structure are position 0 and the positions of structure references (Execute Structure, Conditional Execute Structure, Include Structure, and Conditional Include Structure elements). However it is permissible to set a renderer's *pick start path* (§ 10.2.33) and a search context's *start path* (§ 16.1.1.1) to the element preceding a structure reference since the corresponding operations resume with the element following the path. The positions of element references are not affected by locking a structure and remain unchanged from the time the structure is locked, unless a **PEXDestroyStructures** request causes element reference deletions; the positions of remaining element references decrease by one for each structure reference deleted above them.

Table 12-2 describes the limitations on structure operations when a structure's permission is *StructureLocked*. All other requests perform the same independently of the structure's permission, including requests **PEXChangeStructureReferences** and **PEXDestroyStructures**, and those requests operating strictly on the structure network.

**Table 12-2 The Limitations on Requests Performed on a Structure whose Permission is *StructureLocked***

| Request | Limitations |
|---|---|
| **PEXAccumulateState** | A *StructurePermission* error is issued if *path* includes an invalid position in a locked structure. |
| **PEXCopyElements** | A *StructurePermission* error is issued if either the source or destination structure has permission *StructureLocked*. |
| **PEXCopyStructure** | A *StructurePermission* error is issued if either the source or destination structure has permission *StructureLocked.* |
| **PEXDeleteElements**<br>**PEXDeleteElementsBetweenLabels**<br>**PEXDeleteElementsToLabel** | A *StructurePermission* error is issued. |
| **PEXElementSearch** | A *StructurePermission* error is issued. |
| **PEXEndPickAll** | The element positions of selected primitives in locked structures are undefined.<br>If the last selected primitive is in a locked structure and it is necessary for the renderer to set a *pick start path,* the renderer uses the element preceding the first structure reference in the locked structure beyond that primitive, or the reference to the locked structure if there are no further structure references in the locked structure. |
| **PEXFetchElements** | A *StructurePermission* error is issued. |
| **PEXGetElementInfo** | A *StructurePermission* error is issued. |
| **PEXGetStructureInfo** | The edit mode, number of elements, size of the elements, and element pointer returned are undefined. |
| **PEXPickAll** | The element positions of selected primitives in locked structures are undefined.<br>If the last selected primitive is in a locked structure and it is necessary for the renderer to set a *pick start path,* the renderer uses the element preceding the first structure reference in the locked structure beyond that primitive, or the reference to the locked structure if there are no further structure references in the locked structure.<br>A *StructurePermission* error is issued if the renderer's *start pick path* selects an invalid position in a locked structure. |
| **PEXRenderElements** | A *StructurePermission* error is issued if *path* includes an invalid position in a locked structure. |

**Table 12-2 The Limitations on Requests Performed on a Structure whose Permission is *StructureLocked***

| Request | Limitations |
|---|---|
| **PEXSearchNetwork** | If this request needs to set a *start path* in a locked structure, the element preceding the first structure reference in the locked structure beyond the primitive which met the search criteria is used or the reference to the locked structure if there are no further structure references in the locked structure. The element position of a primitive selected in a locked structure is undefined.<br>A *StructurePermission* error is issued if the *start path* selects an invalid position in a locked structure. |
| **PEXSetEditingMode** | A *StructurePermission* error is issued. |
| **PEXSetElementPointer**<br>**PEXSetElementPointerAtLabel**<br>**PEXSetElementPointerAtPickID** | A *StructurePermission* error is issued. |
| **PEXSetStructurePermission** | A *StructurePermission* error is issued. |
| **PEXStoreElements** | A *StructurePermission* error is issued. |

A structure’s *permission* is changed with the **PEXSetStructurePermission** request. Structures with permission *StructureWriteOnly* or *StructureLocked* cannot be given a permission of *StructureReadWrite*. Structures whose permission is *StructureLocked* cannot have their permission changed.

Modifying the permission of a structure has no effect on any other structure. For example, locking a structure which references non-locked structures will not cause the referenced structures to have their permissions modified.

## 12.2 Structure Requests

A structure is an X11 resource and carries all the responsibilities and access rights of X11 resources. The requests that operate on structures are defined in this section.

### 12.2.1 Element Positions and Ranges

Several structure requests have ELEMENT_POS (§3.7.7.46) parameters identifying an element position within a structure. These parameters are called *whence* and *offset*. The *whence* parameter indicates a reference point within the structure, either element position 0 (*Beginning*), the position of the last element in the structure (*End*), or the position of the current element pointer (*Current*). The value of *offset* is added to this reference position to determine the element position used in the request. *offset* can be positive or negative. If adding *offset* to the position indicated by *whence* yields a position less than 0, then position 0 is selected. If it yields a position beyond that of the last element in the structure, then the position of the last element in the structure is selected.

Requests that operate on ranges of elements use the ELEMENT_RANGE (§3.7.7.47) data type, which contains two *whence-offset* pairs. One pair identifies the first position of the range and the other identifies the last position. The first and last position are included in the range. The rules above for computing an element position are applied independently to the two *whence-offset* pairs of an ELEMENT_RANGE. There is no implied ordering of the two pairs; each may designate either the first or last position of the range.

### 12.2.2 Change Structure References

5.0

**Name**

**PEXChangeStructureReferences**

**Request**

| | |
|---|---|
| *old_s_id*: | STRUCTURE_ID |
| *new_s_id*: | STRUCTURE_ID |

**Errors**

Structure
*old_s_id* or *new_s_id* is invalid.

**Description**

This request modifies all structure referencing elements in the server's structures which reference the structure identified by *old_s_id*. The references are modified to reference the structure identified by *new_s_id* instead. Existing references to *new_s_id* are not affected.

If *new_s_id* is already posted, its posting is not modified. On PHIGS workstation resources where *old_s_id* is posted but *new_s_id* is not, a posting is added for *new_s_id* at the same priority as *old_s_id*. *old_s_id* is unposted on all PHIGS workstation resources where it is posted.

If there are references to the *old_s_id* and *new_s_id* does not exist, an error is returned and no action is taken.

**Encoding: PEXChangeStructureReferences**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 48 | PEX opcode |
| 2 | 3 | request length |
| 4 | STRUCTURE_ID | old structure ID |
| 4 | STRUCTURE_ID | new structure ID |

### 12.2.3 Copy Elements

5.0

**Name**

**PEXCopyElements**

**Request**

| | |
|---|---|
| *src_s_id*: | STRUCTURE_ID |
| *src_range*: | ELEMENT_RANGE |
| *dest_s_id*: | STRUCTURE_ID |
| *dest_position*: | ELEMENT_POS |

**Errors**

Structure
*src_s_id* or *dest_s_id* is invalid.

Value
*src_range* or *dest_position* contains an invalid whence value.

StructurePermission
Elements in a structure whose permission is *StructureWriteOnly* structure cannot be copied to a structure whose permission is *StructureReadWrite*. The structure also cannot be copied if either source of destination structure has a permission of *StructureLocked*.

**Description**

This request copies elements in *src_range* from the structure specified by *src_s_id* to the position immediately following *dest_position* in the structure specified by *dest_s_id*. The elements are inserted in the destination structure; the structure's edit mode does not apply. After the elements are inserted, the destination structure's element pointer is set to the position of the last element inserted.

If *src_s_id* and *dest_s_id* are the same, the operation performs as if the indicated range is copied to a temporary location, and then inserted relative to the destination position. If position 0 is given as the start of the element range, the implicit element at position 0 is ignored and the copy commences with the first element.

**Encoding: PEXCopyElements**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 47 | PEX opcode |
| 2 | 9 | request length |
| 4 | STRUCTURE_ID | source structure ID |
| 16 | ELEMENT_RANGE | source range (§ 12.2.1) |
| 4 | STRUCTURE_ID | destination structure ID |
| 8 | ELEMENT_POS | destination position |

### 12.2.4 Copy Structure

5.0

**Name**

**PEXCopyStructure**

**Request**

| | |
|---|---|
| *src_s_id*: | STRUCTURE_ID |
| *dest_s_id*: | STRUCTURE_ID |

**Errors**

Structure
*src_s_id* or *dest_s_id* is invalid.

StructurePermission
Elements in a structure whose permission is *StructureWriteOnly* cannot be copied to a structure whose permission is *StructureReadWrite*. Neither the source or destination structure can have a permission of *StructureLocked*.

**Description**

This request deletes all elements in the structure identified by *dest_s_id* and then inserts into that structure all the elements in the structure identified by *src_s_id*. The edit mode and element pointer of *dest_s_id* are then set to that of *src_s_id*. The permission of the destination structure is not modified by this request.

**Encoding: PEXCopyStructure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 31 | PEX opcode |
| 2 | 3 | request length |
| 4 | STRUCTURE_ID | source structure ID |
| 4 | STRUCTURE_ID | destination structure ID |

### 12.2.5 Create Structure

5.0

**Name**

**PEXCreateStructure**

**Request**

*s_id*: STRUCTURE_ID

**Errors**

Alloc
The server failed to allocate the structure.

IDChoice
The resource ID given in *s_id* is already in use or not in the range assigned to the client.

**Description**

This request creates a structure resource with the specified *s_id*. The structures edit mode is set to *Insert*. Its element pointer is set to 0. Its permission is *StructureReadWrite*.

**Encoding: PEXCreateStructure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 30 | PEX opcode |
| 2 | 2 | request length |
| 4 | STRUCTURE_ID | structure ID |

### 12.2.6 Delete Elements

**Name**

**PEXDeleteElements**

**Request**

*s_id*: STRUCTURE_ID
*range*: ELEMENT_RANGE

**Errors**

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

Value
*range* contains an invalid whence value.

**Description**

This request deletes elements in *range* (§ 12.2.1 ) from the structure identified by *s_id*. Element position 0 may be in the range, in which case no element is deleted for that position. After the deletion the structure's element pointer is set to the position immediately preceding the range of deleted elements.

**Encoding: PEXDeleteElements**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 44 | PEX opcode |
| 2 | 6 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 16 | ELEMENT_RANGE | element range |

### 12.2.7 Delete Elements Between Labels

5.0

**Name**

**PEXDeleteElementsBetweenLabels**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *label1*: | INT32 |
| *label2*: | INT32 |

**Errors**

Structure
*s_id* is invalid.

Label
One or both of the labels were not found.

StructurePermission
The structure's permission is *StructureLocked*.

**Description**

This request deletes all elements between the next instance of *label1* and the subsequent instance of *label2* in the structure identified by *s_id*. A search for *label1* is first performed starting from the current element pointer position plus one. A search for *label2* is then performed, starting at the element immediately succeeding *label1*. The elements between the two labels are then deleted. The two label elements themselves are not deleted. After the deletion, the structure's element pointer is set to point to *label1*.

If either of the two specified labels is not found between starting point of the search and the end of the structure, no deletion occurs and the structure's element pointer is left unchanged.

**Encoding: PEXDeleteElementsBetweenLabels**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 46 | PEX opcode |
| 2 | 4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 4 | INT32 | label 1 |
| 4 | INT32 | label 2 |

### 12.2.8 Delete Elements To Label

5.0

**Name**

**PEXDeleteElementsToLabel**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *position*: | ELEMENT_POS |
| label: | INT32 |

**Errors**

Label
The label was not found.

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

Value
*position* contains an invalid whence value.

**Description**

The request deletes all elements between the element position indicated by *position* and the next instance of *label* in the structure identified by *s_id*. The label element and the element indicated by *position* are not deleted. After the deletion, the element pointer is set to *position*. If *label* is not found, no deletion occurs and the structure's element pointer is left unchanged.

**Encoding: PEXDeleteElementsToLabel**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 45 | PEX opcode |
| 2 | 5 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 8 | ELEMENT_POS | element position |
| 4 | INT32 | label |

### 12.2.9 Destroy Structures

5.0

**Name**

**PEXDestroyStructures**

**Request**

*list*: LISTofSTRUCTURE_ID

**Errors**

Structure
One or more of the structures identified by *list* is invalid.

**Description**

This request destroys the structures indicated in *list*. Structure elements throughout the server's structure store, such as elements of type Execute Structure, Include Structure, Conditional Execute Structure, and Conditional Include Structure, that refer to a structure in *list* are also deleted. Any of the destroyed structures posted to a PHIGS workstation resource is unposted.

Paths in search contexts that contain the ID of a structure in *list* will not be modified and may still be queried, but the **PEXSearchNetwork** and **PEXUpdatePickMeasures** requests generate an error if a path with the ID of a destroyed structure is used.

The element pointer of structures that have elements deleted during this request are updated to remain pointing to the same element they pointed to before the deletion. If the element pointer before the deletion contains the position of a deleted element, it is set to the element immediately preceding the deleted element.

**Encoding: PEXDestroyStructures**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 32 | PEX opcode |
| 2 | 2+*n* | request length |
| 4 | CARD32 | number of structure IDs (*n*) |
| 4**n* | LISTofSTRUCTURE_ID | list of structure IDs |

### 12.2.10 Element Search

5.0

**Name**

**PEXElementSearch**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *position*: | ELEMENT_POS |
| *direction*: | {*Forward*, *Backward*} |
| *incl*: | LISTofELEMENT_TYPE |
| *excl*: | LISTofELEMENT_TYPE |

**Reply**

| | |
|---|---|
| *status*: | {*Found*, *NotFound*} |
| *found_offset*: | CARD32 |

**Errors**

StructurePermission
: The structure's permission is *StructureLocked*.

Structure
: *s_id* is invalid.

Value
: *position* or *direction* contains an invalid value.

**Description**

This request searches for the first occurrence of the specified element type in the structure identified by *s_id*. An element is selected if its element type is not contained in *excl* and its element type is included in *incl*. An element type of *OCAll* in *incl* or *excl* causes all element types, including *OCNil*, to be either included or excluded, respectively. If an element type is in both *incl* and *excl*, it is excluded. The implicit element at element position 0 of a structure is of type *OCNil*.

The search terminates if a match is found or if the limits of the structure are reached. The search begins at the starting position indicated by *position* (the element at that position is included in the search), and proceeds either forward in the structure or backward, depending on *direction*. The search does not descend into structures executed or included by *s_id*. If a match is found, a status of *Found* is returned and the element position of the matching element is returned in *found_offset*. If the search is unsuccessful, a status of *NotFound* is returned and 0 is returned in *found_offset*.

The structure's element pointer is not modified by this request.

**Encoding: PEXElementSearch**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 42 | PEX opcode |
| 2 | 7+(2**i*+2**e*+*p*+*p2*)/4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 8 | ELEMENT_POS | element position |
| 4 | CARD32 | direction |
| | 0 *Forward* 5.0 | |
| | 1 *Backward* 5.0 | |
| 4 | CARD32 | number of element types in inclusion list (*i*) |
| 4 | CARD32 | number of element types in exclusion list (*e*) |
| 2**i* | LISTofELEMENT_TYPE | inclusion list |
| *p* | | *p*=pad(2**i*) |
| 2**e* | LISTofELEMENT_TYPE | exclusion list |
| *p2* | | *p2*=pad(2**e*) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 0 | reply length |
| 2 | CARD16 | status |
| | 1 *NotFound* 5.0 | |
| | 2 *Found* 5.0 | |
| 2 | | unused |
| 4 | CARD32 | found element position |
| 16 | | unused |

### 12.2.11 Fetch Elements

5.0

**Name**

**PEXFetchElements**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *s_id*: | STRUCTURE_ID |
| *range*: | ELEMENT_RANGE |

**Reply**

| | |
|---|---|
| *elements*: | LISTofOUTPUT_CMD |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

StructurePermission
The structure's permission is *StructureWriteOnly* or *StructureLocked*.

Structure
*s_id* is invalid.

Value
*range* contains an invalid whence value.

**Description**

This request returns the contents of the structure elements indicated in *range* (§ 12.2.1) from the structure identified by *s_id*. The contents of each structure element are returned in the same format as described in Chapter 6. Floating point values in *elements* are returned in the floating point format specified in *fp_format*. No information is returned for inquiries of element position zero.

The structure's element pointer is not modified by this request.

**Encoding: PEXFetchElements**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 38 | PEX opcode |
| 2 | 7 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | STRUCTURE_ID | structure ID |
| 16 | ELEMENT_RANGE | element range |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *s*/4 | reply length |
| 4 | CARD32 | number of elements returned |
| 20 | | unused |
| s | LISTofOUTPUT_CMD | element contents (§3.7.7.97) |

### 12.2.12 Get Ancestors

**Name**

**PEXGetAncestors**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *path_part*: | {*TopPart*, *BottomPart*} |
| *path_depth*: | CARD32 |

**Reply**

| | |
|---|---|
| *paths*: | LISTofLISTofELEMENT_REF |

**Errors**

Structure
*s_id* is invalid.

Value
*path_part* contains an invalid value.

**Description**

This request returns unique paths in the structure hierarchy that lead to *s_id*. Paths are returned as a list of structure id/element offset pairs that are in the order from the root structure down to *s_id*. For full paths, the last entry in each path would have the structure ID equal to *s_id* and the offset equal to zero. The *path_depth* indicates the maximum length (number of structure id/element offset pairs) of each path, except that *path_depth* = 0 means that the full path is to be returned. The *path_part* indicates whether the head or the tail of each path is to be returned. If *path_part* = *TopPart*, only the first *path_depth* structure id/element offset pairs in each path are returned. If *path_part* = *BottomPart*, only the last *path_depth* structure id/element offset pairs in each path will be returned. Only unique paths are returned (i.e., there are no duplicates in the list of returned paths).

For example, specifying *path_depth* = 0 causes all paths leading to *s_id* to be returned. Specifying *path_depth* = 1 and *path_part* = *TopPart* causes all the structures at the top of each structure network containing *s_id* to be returned. Specifying *path_depth* = 1 and *path_part* = *BottomPart* allows a client to determine whether or not *s_id* is referenced. Specifying *path_depth* = 2 and *path_part* = *BottomPart* allows a client to determine the number of immediate ancestors for *s_id* as well as returning their resource IDs.

**Encoding: PEXGetAncestors**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 36 | PEX opcode |
| 2 | 4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 2 | CARD16 | path part |
| | 0 *TopPart* 5.0 | |
| | 1 *BottomPart* 5.0 | |
| 2 | | unused |
| 4 | CARD32 | path depth |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $n$+2*$s$ | reply length |
| 12 | | unused |
| 4 | CARD32 | number of paths ($n$) |
| 8 | | unused |
| 4*$n$+8*$s$[†1] | LISTofLISTofELEMENT_REF | paths (§3.7.7.48) |

†1) $s$ is the total number of ELEMENT_REF entries returned.

### 12.2.13 Get Descendants

5.0

**Name**

**PEXGetDescendants**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *path_part*: | {*TopPart*, *BottomPart*} |
| *path_depth*: | CARD32 |

**Reply**

| | |
|---|---|
| *paths*: | LISTofLISTofELEMENT_REF |

**Errors**

Structure
*s_id* is invalid.

Value
*path_part* contains an invalid value.

**Description**

This request returns unique paths in the structure hierarchy from *s_id* to leaf nodes in the hierarchy. Paths are returned as a list of structure id/element offset pairs that are in the order from *s_id* down to the leaf nodes. For full paths, the first entry in each path would have the structure ID equal to *s_id* and the offset equal to the offset of the element that references a descendant in the hierarchy, and the last entry would have the structure ID of a leaf node in the structure hierarchy and an offset of zero. The *path_depth* indicates the maximum length (number of structure id/element offset pairs) of each path, except that *path_depth* = 0 means that the full path is to be returned. The *path_part* indicates whether the head or the tail of each path is to be returned. If *path_part* = *TopPart*, only the first *path_depth* structure id/element offset pairs in each path are returned. If *path_part* = *BottomPart*, only the last *path_depth* structure id/element offset pairs in each path are returned. Only unique paths will be returned (i.e., there will be no duplicates in the list of returned paths).

For example, specifying *path_depth* = 0 causes all paths leading from *s_id* to leaf nodes to be returned. Specifying *path_depth* = 1 and *path_part* = *TopPart* causes all the direct references to other structures in *s_id* to be returned. Specifying *path_depth* = 1 and *path_part* = *BottomPart* could be used to determine the leaf nodes for the structure hierarchy rooted at *s_id*.

**Encoding: PEXGetDescendants**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 37 | PEX opcode |
| 2 | 4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 2 | CARD16 | path part |
| | 0 *TopPart* 5.0 | |
| | 1 *BottomPart* 5.0 | |
| 2 | | unused |
| 4 | CARD32 | path depth |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $n$+2*$s$ | reply length |
| 12 | | unused |
| 4 | CARD32 | number of paths ($n$) |
| 8 | | unused |
| 4*$n$+8*$s$ †1 | LISTofLISTofELEMENT_REF | paths (§3.7.7.48) |

†1) $s$ is the total number of ELEMENT_REF entries returned.

### 12.2.14 Get Element Info

5.0

**Name**

**PEXGetElementInfo**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *s_id*: | STRUCTURE_ID |
| *range*: | ELEMENT_RANGE |

**Reply**

| | |
|---|---|
| *info*: | LISTofELEMENT_INFO |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

Value
*range* contains an invalid value.

**Description**

This request returns information about the elements indicated by *range* in the structure identified by *s_id*. Each element's element type is returned, as well as the length of the element in units of four bytes. (The sum of the lengths of all elements in the structure is equal to the length that can be inquired with the **PEXGetStructureInfo** request.) The element lengths are computed for the floating-point type indicated by *fp_format*. The length returned for each element is undefined if the structure's permission is *StructureWriteOnly*. No information is returned for inquiry of element position 0.

Just as output command primitives can be encoded with a 32-bit length, so the returned length also follows the same convention where the 16-bit length may be given as 0, indicating that the true length of the output command is held in the 32-bit word (an unsigned integer) immediately following the 16-bit length.

The structure's element pointer is not modified by this request.

**Encoding: PEXGetElementInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 34 | PEX opcode |
| 2 | 7 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | STRUCTURE_ID | structure ID |
| 16 | ELEMENT_RANGE | element range |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $n+k*n/4$ | reply length |
| 4 | CARD32 | number of elements whose info is returned ($n$) |
| 20 | | unused |
| $(4+k)*n$ | LISTofELEMENT_INFO | element info (§12.2.14.1) |

### 12.2.14.1 Type ELEMENT_INFO

ELEMENT_INFO : [type : ELEMENT_TYPE,
length : CARD16]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | ELEMENT_TYPE | element type (§3.7.7.49) |
| 2 | CARD16 | length |
| $k^{\dagger 1}$ | CARD32 | 32-bit length |

†1) $k$ = ((length == 0) ? 4 : 0)

### 12.2.15 Get Structure Info

**Name**

**PEXGetStructureInfo**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *s_id*: | STRUCTURE_ID |
| *item_mask*: | BITMASK_SHORT |

**Reply**

| | |
|---|---|
| *edit_mode*: | {*StructureInsert*, *StructureReplace*} |
| *element_ptr*: | CARD32 |
| *num_elements*: | CARD32 |
| *total_length*: | CARD32 |
| *has_refs*: | BOOLEAN16 |
| *permission*: | {*StructureReadWrite*, *StructureWriteOnly*, *StructureLocked*} |

**Errors**

FloatingPointFormat
: The floating-point format indicated by *fp_format* is not supported.

Structure
: *s_id* is invalid.

Value
: A non-defined bit in *item_mask* is set.

**Description**

This request returns the information indicated by *item_mask* from the structure specified by *s_id*. Items indicated by *item_mask* have valid values returned, and the remaining values are returned as undefined values.

The current value of the structure's editing mode attribute will be returned in *edit_mode*. The number of structure elements in the structure is returned in *num_elements*. The total number of four-byte units necessary to store all elements of the structure is returned in *total_length*. (This provides the information needed by a client to allocate memory prior to a **PEXFetchElements** request.) The total length is computed using the floating-point format indicated in *fp_format*. If there are any Execute Structure, Conditional Execute Structure, Include Structure, or Conditional Include Structure elements that refer to *s_id* in other structures, then *has_refs* is *True*; otherwise *has_refs* is *False*. The current structure permission is returned in *permission*.

If the structure's permission is *StructureWriteOnly*, *total_length* has an undefined value. If the structure's permission is *StructureLocked*, *edit_mode, element_ptr*, *num_elements,* and *total_length* have undefined values.

**Encoding: PEXGetStructureInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 33 | PEX opcode |
| 2 | 3 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | BITMASK_SHORT | item mask |
| | 0x0001 element pointer 5.0 | |
| | 0x0002 number of elements 5.0 | |
| | 0x0004 total length 5.0 | |
| | 0x0008 has references 5.0 | |
| | 0x0010 edit mode 5.0 | |
| | 0x0020 permission 5.2 | |
| 4 | STRUCTURE_ID | structure ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 0 | reply length |
| 2 | CARD16 | editing mode |
| | 0 *StructureInsert* 5.0 | |
| | 1 *StructureReplace* 5.0 | |
| 2 | | unused |
| 4 | CARD32 | element pointer |
| 4 | CARD32 | number of elements |
| 4 | CARD32 | total length |
| 2 | BOOLEAN16 | has references |
| 2 | CARD16 | permission |
| | 0 *StructureReadWrite* 5.2 | |
| | 1 *StructureWriteOnly* 5.2 | |
| | 2 *StructureLocked* 5.2 | |
| 4 | | unused |

### 12.2.16 Get Structures In Network 5.0

**Name**

**PEXGetStructuresInNetwork**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *which*: | {*All*, *NoCrossRefs*} |

**Reply**

| | |
|---|---|
| *structures*: | LISTofSTRUCTURE_ID |

**Errors**

Structure
: *s_id* is invalid.

Value
: The value in *which* is not recognized.

**Description**

This request returns a list of unique structure IDs of structures referenced in the structure network rooted at *s_id*. If *which* is *All*, the ID of all structures in the network are returned. If *which* is *NoCrossRefs*, only the IDs of structures not executed or included from outside the indicated network are returned. *s_id* is included for both values of *which*.

**Encoding: PEXGetStructuresInNetwork**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 35 | PEX opcode |
| 2 | 3 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 2 | CARD16 | which |
| | 0 *All* 5.0 | |
| | 1 *NoCrossRefs* 5.0 | |
| 2 | | unused |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *n* | reply length |
| 8 | | unused |
| 4 | CARD32 | number of structures (*n*) |
| 12 | | unused |
| 4**n* | LISTofSTRUCTURE_ID | structure IDs |

### 12.2.17 Set Editing Mode 5.0

**Name**

**PEXSetEditingMode**

**Request**

*s_id*: STRUCTURE_ID
*mode*: {*StructureInsert*, *StructureReplace*}

**Errors**

StructurePermission
The structure’s permission is *StructureLocked*.

Structure
*s_id* is invalid.

Value
The value in *mode* is not recognized.

**Description**

This request sets the editing mode of the structure identified by *s_id* to the value given in *mode*. See request **PEXStoreElements** (§ 12.2.22) for a description of the effects of the editing modes.

**Encoding: PEXSetEditingMode**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 39 | PEX opcode |
| 2 | 3 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 2 | CARD16 | mode |
| | 0 *StructureInsert* 5.0 | |
| | 1 *StructureReplace* 5.0 | |
| 2 | | unused |

### 12.2.18 Set Element Pointer

**Name**

**PEXSetElementPointer**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *position*: | ELEMENT_POS |

**Errors**

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

Value
The whence value in *position* is invalid.

**Description**

This request sets the element pointer of the structure indicated by *s_id* to the element position given by *position*.

**Encoding: PEXSetElementPointer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 40 | PEX opcode |
| 2 | 4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 8 | ELEMENT_POS | element position |

### 12.2.19 Set Element Pointer At Label

**Name**

**PEXSetElementPointerAtLabel**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *label*: | INT32 |
| *offset*: | INT32 |

**Errors**

Label
The label does not exist between the current element pointer and the end of the structure.

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

**Description**

This request sets the element pointer of the structure indicated by *s_id*. A search is conducted for a Label element with the value indicated by *label*, beginning with the element immediately succeeding the element indicated by the structure's element pointer. The search is non-descending, that is, it does not include any structures referenced by *s_id*. If the label is found, the element pointer is set to the position computed by adding *offset*, which may have any integer value, to the position of the label. *offset* may be positive or negative. If the computed position is less than 0, the element pointer is set to 0. If the computed position is beyond that of the last element in the structure, the element pointer is set to the position of the last element in the structure. If the specified label is not found, an error is generated and the structure's element pointer is not changed.

**Encoding: PEXSetElementPointerAtLabel**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 41 | PEX opcode |
| 2 | 4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 4 | INT32 | label |
| 4 | INT32 | offset |

### 12.2.20 Set Element Pointer At Pick ID

**Name**

**PEXSetElementPointerAtPickID**

**Request**

| | |
|---|---|
| *s_id*: | STRUCTURE_ID |
| *pick_id*: | INT32 |
| *offset*: | INT32 |

**Errors**

Label
The pick ID does not exist between the current element pointer and the end of the structure.

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

**Description**

This request sets the element pointer of the structure indicated by *s_id*. A search is conducted for a Set Pick Identifier element with the value indicated by *pick_id*, beginning with the element immediately succeeding the element indicated by the structure's element pointer. The search is non-descending, that is, it does not include any structures referenced by *s_id*. If the element is found, the element pointer is set to the position computed by adding *offset*, which may have any integer value, to the position of that element. *offset* may be positive or negative. If the computed position is less than 0, the element pointer is set to 0. If the computed position is beyond that of the last element in the structure, the element pointer is set to the position of the last element in the structure. If the specified pick ID is not found, an error is generated and the structure's element pointer is not changed.

**Encoding: PEXSetElementPointerAtPickID**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 122 | PEX opcode |
| 2 | 4 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 4 | INT32 | pick ID |
| 4 | INT32 | offset |

### 12.2.21 Set Structure Permission

**Name**

**PEXSetStructurePermission**

**Request**

*s_id*: STRUCTURE_ID
*permission*: {*StructureWriteOnly*, *StructureLocked*}

**Errors**

Structure
*s_id* is invalid.

Value
The value in *permission* is invalid.

StructurePermission
Structures with permission *StructureWriteOnly* can only be given a permission of *StructureLocked*. Structures whose permission is *StructureLocked* cannot have their permission reset.

**Description**

This request sets the permission (§ 12.1.1.3) of the structure identified by *s_id* to the value given in *permission*.

**Encoding: PEXSetStructurePermission**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 123 | PEX opcode |
| 2 | 3 | request length |
| 4 | STRUCTURE_ID | structure ID |
| 2 | CARD16 | permission |
| | 1 *StructureWriteOnly* | |
| | 2 *StructureLocked* | |
| 2 | | unused |

### 12.2.22 Store Elements

5.0

**Name**

**PEXStoreElements**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *s_id*: | STRUCTURE_ID |
| *output_cmds*: | LISTofOUTPUT_CMD |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

OutputCommand
One or more of the output commands contains an illegal value.

StructurePermission
The structure's permission is *StructureLocked*.

Structure
*s_id* is invalid.

**Description**

This request stores output commands in the structure identified by *s_id*. If the *editing mode* of the structure is *StructureInsert*, the output commands are inserted as structure elements between the element indicated by the structure's element pointer and its succeeding element. If the editing mode is *StructureReplace*, the element indicated by the structure's element pointer and its succeeding *N–1* elements (or all of the succeeding elements if there are less than *N–1* elements from that position in the structure) are deleted and replaced by the *N* output commands.

The structure's element pointer is set to the position of the last element inserted or replaced by this request. Note when the *editing mode* is *StructureReplace*, for the same set of output commands, a single **PEXStoreElements** request is not semantically equivalent to buffering the output commands across multiple **PEXStoreElements** requests.

Floating point values in *output_cmds* are in the floating point format specified by *fp_format*.

If an error is found in one of the output commands, all output commands preceding it in *output_cmds* are stored in the structure, but the erroneous output command and all those subsequent to it in *output_cmds* are not stored and an error is generated.

**Encoding: PEXStoreElements**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 43 | PEX opcode |
| 2 | 4+*s*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | STRUCTURE_ID | structure ID |
| 4 | CARD32 | number of output commands (*n*) |
| s | LISTofOUTPUT_CMD | output commands (§3.7.7.97) |

# 13 Fonts

## 13.1 PEX Fonts

The PEX font resource is an X11 resource representing a text font supported by PEX. It is analogous to the X font resource, which represents text fonts supported by the X server. It is possible for the same font to be supported by both X and PEX or just one. Requests **PEXListFonts** and **PEXListFontsWithInfo** indicate which fonts are supported by PEX.

PEX uses the X11 font path to find the font files. X11 request **SetFontPath** is used to set the font path. X11 request **GetFontPath** is used to query the font path. Request **PEXOpenFont** is used to create a PEX font resource. The association between a PEX font and its resource identifier is deleted with request **PEXCloseFont**.

The PEX font resource requests are listed in Table 13-1 .

**Table 13-1 . Text Font Resource Management Requests**

| Request | Description |
|---|---|
| **PEXOpenFont** | Load a PEX font. |
| **PEXCloseFont** | Free a PEX font. |
| **PEXQueryFont** | Retrieve a PEX font's attributes. |
| **PEXListFonts** | Retrieve the supported PEX fonts. |
| **PEXListFontsWithInfo** | Retrieve the supported PEX fonts and their attributes. |
| **PEXQueryTextExtents** | Retrieve the extents of the specified strings. |

### 13.1.1 Font Naming

In PEX 5.1 and earlier revisions, there were no common naming requirements. In PEX 5.2, however, XLFD(X Logical Font Description) naming conventions are required. This includes use of patterns with wildcards. For backwards compatibility, some server implementations may provide access to fonts via multiple names, one using the XLFD name and one or more from previous implementation heritage. This is permitted, as long as every font has at least a name which abides by the XLFD conventions.

Each server is required to support at least one font which supports the character set ISO 8859/1 (Latin1) and is of at least medium weight. The XLFD name for this font must match the following XLFD match pattern:

-*-*-*-*-*-*-*-*-*-*-*-*-ISO8859-1

### 13.1.2 Font Information

The information listed in Table 13-2 is defined for each PEX font. Request **PEXQueryFont** returns this information for a given PEX font resource. Request **PEXListFontsWithInfo** returns this information for each font supported by PEX.

**Table 13-2 . PEX_FONTINFO - PEX Font Information**

| Field | Data Type | Description |
|---|---|---|
| *first glyph* | CARD32 | The value of the first glyph in the font. 5.0 |
| *last glyph* | CARD32 | The value of the last glyph in the font. 5.0 |
| *default glyph* | CARD32 | The value of the font’s default glyph, the glyph displayed when an undefined or nonexistent glyph is requested. *default glyph* may indicate a non-existent glyph, in which case nothing is displayed when the default glyph is used. 5.0 |
| *all glyphs exist* | BOOLEAN | *True* if glyphs exist in the font for all values between *first glyph* and *last glyph*. *False* if not all those glyphs exist. 5.0 |
| *stroke font* | BOOLEAN | *True* if the font supports PEX text precision *Stroke (§6.9.95)*. 5.0 |
| *properties* | LISTofPEX_FONTPROP | Any properties defined for the font at the time of inquiry (§13.1.2.1). 5.0 |

**Encoding: PEX_FONTINFO**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | CARD32 | first glyph |
| 4 | CARD32 | last glyph |
| 4 | CARD32 | default glyph |
| 1 | BOOLEAN | all glyphs exist |
| 1 | BOOLEAN | stroke font |
| 2 | | unused |
| 4 | CARD32 | number of properties ($n$) |
| 8*$n$ | LISTofPEX_FONTPROP | properties |

#### 13.1.2.1 Font Properties

In PEX 5.1 and earlier revisions, properties were optional, but recommended. Given the requirement to meet XLFD conventions, the XLFD font properties should be used, if any properties are provided to give additional information about fonts. However, XLFD font properties are optional. The data type and encoding for font properties is:

PEX_FONTPROP : [name : ATOM,
value : CARD32]

**Encoding:**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | ATOM | name |
| 4 | CARD32 | value |

#### 13.1.2.2 Text Precision

Although each PEX font must support at least *String* and *Char* precision (§6.9.95), it does not have to support text precision *Stroke*. The *stroke font* field of PEX_FONTINFO indicates whether the font supports the *Stroke* precision, which implies the font is scalable and rotatable.

The text precision determines to what minimum extent each of the text attributes must be realized (§6.9.95). The text attributes are defined in the Annotation, Text, and Annotation And Text attribute sets (§6.5). A font cannot be a PEX font unless it can realize the PEX text attributes to the extent required by the text precisions supported by that font.

## 13.2 Font Requests

The requests that operate on PEX font resources are defined in this section.

### 13.2.1 Close PEX Font

5.0

**Name**

**PEXCloseFont**

**Request**

*font_id*: PEX_FONT_ID

**Errors**

PEXFont
*font_id* is not associated with a font resource.

**Description**

This request deletes the association between *font_id* and the font. The font is freed when no other resource refers to it.

**Encoding: PEXCloseFont**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 89 | PEX opcode |
| 2 | 2 | request length |
| 4 | PEX_FONT_ID | font ID |

### 13.2.2 List PEX Fonts

5.0

**Name**

**PEXListFonts**

**Request**

*pattern*: STRING
*max_names*: CARD16

**Reply**

*font_names*: LISTofSTRING

**Errors**

None

**Description**

Performs the same operations as the X11 request **ListFonts**, but returns only the names of fonts supported by PEX. (The returned list may contain fonts also returned by the X11 **ListFonts** request.)

The fonts returned are those whose name matches *pattern*, an XLFD font name string.

For backwards compatibility, the pattern string can also be any string encoded in ISO Latin-1, including the '?' character (octal value 77), which matches any single character, and the character '*' (octal value 52), which matches any string of characters.

Pattern matching is not case sensitive.

The returned *font_names* are in lower case and encoded in ISO Latin-1. The request returns at most *max_names* fonts.

**Encoding: PEXListFonts**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 91 | PEX opcode |
| 2 | 2+($n$+$p$)/4 | request length |
| 2 | CARD16 | max names |
| 2+$n$+$p$ | STRING_UNALIGNED_PACKED | pattern |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | $r$/4 | reply length |
| 4 | CARD32 | number of strings in list |
| 20 | | unused |
| $r$ | LISTofSTRING_ALIGNED_PACKED | font names |

### 13.2.3 List PEX Fonts with Info

5.0

**Name**

**PEXListFontsWithInfo**

**Request**

| | |
|---|---|
| *pattern:* | STRING |
| *max_names*: | CARD16 |

**Reply**

| | |
|---|---|
| *font_names*: | LISTofSTRING |
| *font_info*: | LISTofPEX_FONTINFO |

**Errors**

None

**Description**

Performs the same operations as the X11 request **ListFontsWithInfo**, but returns information for only fonts that support all PEX attributes. (The returned list may contain fonts also returned by the X11 **ListFonts** request.) This request functions similarly to **PEXListFonts**, except this request returns additional information about each font.

The fonts returned are those whose name matches *pattern*, an XLFD font name string.

For backwards compatibility, the pattern string can also be any string encoded in ISO Latin-1, including the ‘?’ character (octal value 77), which matches any single character, and the character ‘*’ (octal value 52), which matches any string of characters.

Pattern matching is not case sensitive.

The returned *font_names* are in lower case and encoded in ISO Latin-1. The request returns at most *max_names* fonts.

**Encoding: PEXListFontsWithInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 92 | PEX opcode |
| 2 | 3+(*n*+*p*)/4 | request length |
| 2 | | unused |
| 2 | CARD16 | max names |
| 4+*n*+*p* | STRING_ALIGNED_PADDED | pattern |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 1+(*r*+*f*)/4 | reply length |
| 4 | CARD32 | number of strings in list |
| 20 | | unused |
| *r* | LISTofSTRING_ALIGNED_PACKED | font names |
| 4 | CARD32 | number of PEX_FONTINFO in list |
| *f* | LISTofPEX_FONTINFO (§13.1.2) | font info |

### 13.2.4 Open PEX Font

5.0

**Name**

**PEXOpenFont**

**Request**

| | |
|---|---|
| *font_id*: | PEX_FONT_ID |
| *name*: | STRING |

**Errors**

Alloc
The server failed to allocate a PEX font resource.

IDChoice
*font_id* is already in use or not in the range assigned to the client.

PEXFont
*name* does not identify a font supported by PEX.

**Description**

This request associates the resource identifier *font_id* with the font identified by *name*. If the font is not already loaded in the server, PEX loads it.

The font name is an XLFD font name string.

For backwards compatibility, the font name can also be any string encoded in ISO Latin-1.

Pattern matching is not case sensitive. Upper and lower case of the same character are indistinguishable.

PEX fonts are not associated with a particular screen, and can be used with any renderer resource or PHIGS workstation resource.

**Encoding: PEXOpenFont**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 88 | PEX opcode |
| 2 | $3+(n+p)/4$ | request length |
| 4 | PEX_FONT_ID | font ID |
| $4+n+p$ | STRING32 | name |

### 13.2.5 Query PEX Font

5.0

**Name**

**PEXQueryFont**

**Request**

*font_id*: PEX_FONT_ID

**Reply**

*font_info*: PEX_FONTINFO

**Errors**

PEXFont
*font_id* is not associated with a font resource supported by PEX.

**Description**

This request returns the font information for a PEX font.

**Encoding: PEXQueryFont**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 90 | PEX opcode |
| 2 | 2 | request length |
| 4 | PEX_FONT_ID | font ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *f*/4 | reply length |
| 4 | CARD32 | length of PEX_FONTINFO (*f*) |
| 20 | | unused |
| *f* | PEX_FONTINFO (§ 13.1.2) | font info |

### 13.2.6 Query PEX Text Extents

5.0

**Name**

**PEXQueryTextExtents**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *resource_id*: | RESOURCE_ID |
| *font_group*: | TABLE_INDEX |
| *path*: | CARD16 |
| *expansion*: | FLOAT |
| *spacing*: | FLOAT |
| *height*: | FLOAT |
| *alignment*: | TEXT_ALIGNMENT |
| *strings*: | LISTofLISTofMONO_ENCODING |

**Reply**

| | |
|---|---|
| *extents*: | LISTofEXTENT_INFO |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Match
*resource_id* identifies a table of type other than TextFont.

Value
1) One of the text attribute values is illegal or out of range.
2) *resource_id* does not name a valid renderer, PHIGS workstation or lookup table.
3) *resource_id* does not identify a renderer or PHIGS workstation with an associated TextFont table.

**Description**

This request returns information giving the geometric extent and the concatenation point of the text strings given in *strings*. The extent of a string is the smallest rectangle completely surrounding it, in text-local coordinate system units (§6.7.1 and §6.7.35).

The concatenation point is the position to place another string, in text-local coordinates, if the string is to be extended. A suitable modeling transformation to account for the character up vector may still need to be applied by the client.

Stroke precision is assumed.

The extent and concatenation point of each string is computed independently of the other strings in the list.

If *resource_id* identifies a renderer or PHIGS workstation resource, the Text Font table used to resolve the font group indicated by *font_group* is that associated with the renderer or PHIGS workstation. If *resource_id* identifies a text font lookup table, that table is used.

If a font used to compute the extent has no *default glyph*, then undefined glyphs in *strings* have width and height of 0 and therefore do not contribute to the extent or concatenation point. If the character set selected by one of the mono-encodings defining a string is not available, then the default font group is applied to the entire string. The results are implementation-dependent for that mono-encoding, if the character set is not available in the default font group.

**Encoding: PEXQueryTextExtents**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | opcode (X11 assigned) |
| 1 | 93 | | PEX opcode |
| 2 | 6+(3**fp*+*q*)/4 | | request length |
| 2 | FLOAT_FORMAT | | floating-point format |
| 2 | CARD16 | | path |
| | 0 | *PathRight* | |
| | 1 | *PathLeft* | |
| | 2 | *PathUp* | |
| | 3 | *PathDown* | |
| 2 | TABLE_INDEX | | text font index |
| 2 | | | unused |
| 4 | RESOURCE_ID | | resource ID (renderer, WKS or text font table) |
| *fp* | FLOAT | | character expansion |
| *fp* | FLOAT | | character spacing |
| *fp* | FLOAT | | character height |
| 2 | CARD16 | | vertical alignment |
| | 0 | *ValignNormal* | |
| | 1 | *ValignTop* | |
| | 2 | *ValignCap* | |
| | 3 | *ValignHalf* | |
| | 4 | *ValignBase* | |
| | 5 | *ValignBottom* | |
| 2 | CARD16 | | horizontal alignment |
| | 0 | *HalignNormal* | |
| | 1 | *HalignLeft* | |
| | 2 | *HalignCenter* | |
| | 3 | *HalignRight* | |
| 4 | CARD32 | | number of strings |
| *q* | LISTofLISTofMONO_ENCODING | | strings |
| **Reply:** | | | |
| 1 | 1 | | Reply |
| 1 | | | unused |
| 2 | CARD16 | | sequence number |
| 4 | 6**n* | | request length |
| 24 | | | unused |
| 6**fp***n* | LISTofEXTENT_INFO | | extents (length implicit) |

**Encoding: EXTENT_INFO**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2**fp* | COORD_2D | lower-left |
| 2**fp* | COORD_2D | upper-right |
| 2**fp* | COORD_2D | concatenation point |

# 14 Texture Mapping

## 14.1 General

PEX texture maps are X11 resources and carry all the responsibilities and access rights of X11 resources.

Two requests are defined for creating PEX texture maps and one for freeing texture map resources.

**Table 14-1 . Texture Map Requests**

| Request | Purpose |
|---|---|
| **PEXCreateColorMipmapTM** | Create a color mipmap texture resource from client data. |
| **PEXCreateColorMipmapTMFromResources** | Create a color mipmap texture resource from server resource image data. |
| **PEXFreeTextureMap** | Free a texture resource. |

### 14.1.1 Create Color Mipmap Texture Map

**Name**

**PEXCreateColorMipmapTM**

**Request**

| | |
|---|---|
| *fp_format:* | FLOAT_FORMAT |
| *texture_map*: | TEXTURE_MAP_ID |
| *dimension*: | ENUM_TYPE_INDEX |
| *num_levels*: | CARD16 |
| *texel_type*: | ENUM_TYPE_INDEX |
| *texels*: | LISTofTEXEL_ARRAY |

**Errors**

Alloc
The server failed to allocate a texture map resource.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

IDChoice
ID already in use or not in range assigned to client.

Value
1) The texture map dimension given is not valid.
2) The texture map texel type given is not valid.
3) The number of texel arrays given does not match the number of texel arrays expected.
4) The texture map size is required to be a power of two
5) The texture map size is required to be square
6) Texel component is out of range.

**Description**

This request creates a mipmap texture resource for use in modifying the color of rendered primitives. A mipmap is a precomputed area sampling mechanism which fixes the cost of approximating the average color over a large number of texels in a texture map. A mipmap is created as a pyramid of down-sampled maps, or “levels”. Typically each texel in level *n* equals the average of four texels beneath it in level *n*-1. Thus the resolution of each map typically becomes half of the preceding level until the top level, or “pinnacle”, is reached, which typically has one texel representing the average color of the original base level map. So, if multiple levels are defined, an array of texels is defined for each level.

*dimension* indicates the dimension (see Table 14-2 ) of the texture map being created. *texels* provides the actual texel data. *texel_type* indicates the type (see Table 14-3 ) of texel data as well as the components of the newly created texture resource.

**Table 14-2 . Enumerated Type *TMDimension***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | 1D | Texture map is a one-dimensional texture map. |
| 1 | 2D | Texture map is a two-dimensional texture map. |
| 2 | 3D | Texture map is a three-dimensional texture map. |

**Table 14-3 . Enumerated Type *TMTexelType***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *LuminanceFloat* | A texel which is passed as a floating point value [0.0 -> 1.0] representing an intensity. The texel datatype is: FLOAT. |
| 1 | *LuminanceInt8* | A texel which is passed as an unsigned 8-bit integer [0 -> 255] representing an intensity. The texel datatype is: CARD8. |
| 2 | *LuminanceInt16* | A texel which is passed as an unsigned 16-bit integer [0 -> 65535] representing an intensity. The texel datatype is: CARD16. |
| 3 | *LuminanceAlphaFloat* | A texel which is passed as two floating point values [0.0 -> 1.0] representing an intensity and an alpha value. The texel datatype is: FLOAT. |
| 4 | *LuminanceAlphaInt8* | A texel which is passed as two unsigned 8-bit integers [0 -> 255] representing an intensity and an alpha value. The texel datatype is: CARD8. |
| 5 | *LuminanceAlphaInt16* | A texel which is passed as two unsigned 16-bit integers [0 -> 65535] representing an intensity and an alpha value. The texel datatype is: CARD16. |
| 6 | *RGBFloat* | A texel which is passed as three floating point values [0.0 - 1.0] representing the red, green, and blue components of a color. The texel datatype is: FLOAT. |
| 7 | *RGBInt8* | A texel which is passed as three unsigned 8-bit integers [0 - 255] representing the red, green, and blue components of a color. The texel datatype is: CARD8. |
| 8 | *RGBInt16* | A texel which is passed as three unsigned 16-bit integers [0 - 65535] representing the red, green, and blue components of a color. The texel datatype is: CARD16. |
| 9 | *RGBAlphaFloat* | A texel which is passed as four floating point values [0.0 - 1.0] representing the red, green, and blue components of a color and an alpha value. The texel datatype is: FLOAT. |
| 10 | *RGBAlphaInt8* | A texel which is passed as four unsigned 8-bit integers [0 - 255] representing the red, green, and blue components of a color and an alpha value. The texel datatype is: CARD8. |
| 11 | *RGBAlphaInt16* | A texel which is passed as four unsigned 16-bit integers [0 - 65535] representing the red, green, and blue components of a color and an alpha value. The texel datatype is: CARD16. |

The *num_levels* members of *texels* are ordered to define the texture map, starting with the base level. Each member includes the dimension in each direction of that level of the texture map. The texels are assumed to be stored in t0-t1-t2 order, with t0 varying fastest and t2 varying slowest. The texture coordinates are normalized, so the first value corresponds to each texture coordinate at 0.0, and the last value corresponds to each texture coordinate at 1.0.

Inquire *MaxFastTMSize* with request **PEXGetImpDepConstants** to determine whether there is a maximum size in any dimension on the base level which must be respected to achieve optimized implementation. If PEX has such a limit, it treats larger texture maps in an implementation-dependent manner. PEX implementations may have other restrictions on the dimensions of each level of the texture map. Inquire *PowerOfTwoTMRequired* with **PEXGetImpDepConstants** to determine if each dimension must be a power of two. Inquire *SquareTMRequired* with the same request to determine if all the dimensions on each level must be equal to one another.

Floating point values will be in the floating point format specified in *fp_format*.

**Encoding: PEXCreateColorMipmapTM**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 124 | PEX opcode |
| 2 | 4+*ts*/4 | request length |
| 2 | FLOAT_FORMAT | floating point format |
| 2 | ENUM_TYPE_INDEX | texture map dimension (Table 14-2 ) |
| 4 | TEXTURE_MAP_ID | texture map resource ID |
| 2 | CARD16 | number of levels (*m*) |
| 2 | ENUM_TYPE_INDEX | texel type (Table 14-3 ) |
| *ts* | LISTofTEXEL_ARRAY | texel arrays (*ts* is the sum total, in bytes, of all texels and texel array counts in the *m* texel arrays. See §14.1.1.1) |

#### 14.1.1.1 TEXEL_ARRAY Encoding

**Encoding: TEXEL_ARRAY**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *d* | TEXEL_ARRAY_COUNTS | For texture map *dimension = 1D:*<br>level dimension *c0* in direction *t0,*<br>For texture map *dimension = 2D:*<br>level dimensions *c0* and *c1* in directions *t0* and *t1* respectively,<br>For texture map *dimension = 3D:*<br>level dimension *c0, c1*, and *c2* in directions *t0, t1,* and *t2* respectively.<br>(The encoding depends upon the dimension; see §14.1.1.2)<br>*d* = (*dimension* == *3D*) ? 8 : 4 |
| *n* | LISTofTEXEL | The texels for a single mipmap level.<br>(The encoding depends upon the texel type; see §14.1.1.3).<br>*c* = (dimension == *1D*) ? *c0* :<br>(dimension == *2D*) ? *c0*c1*<br>: *c0*c1*c2*))<br>*n* = (*texel_type* == *LuminanceInt8*) ? *c* :<br>(*texel_type* == (*LuminanceInt16* \| *LuminanceAlphaInt8*) ? 2**c* :<br>(*texel_type* == *RGBInt8*) ? 3*c :<br>(*texel_type* == *LuminanceAlphaInt16* \| *RGBAlphaInt8*) ? 4**c :*<br>(*texel_type* == *RGBInt16*) ? 6**c* :<br>(*texel_type* == *RGBAlphaInt16*) ? 8**c* :<br>(*texel_type* == *LuminanceFloat*) ? *c*fp* :<br>(*texel_type* == *LuminanceAlphaFloat*) ? 2**c*fp* :<br>(*texel_type* == *RGBFloat*) ? 3**c*fp* :<br>(*texel_type* == *RGBAlphaFloat*) ? 4**c*fp* |
| *p* | | *p*=pad(*n*) |

#### 14.1.1.2 TEXEL_ARRAY_COUNTS

**Encoding: TEXEL_ARRAY_COUNTS for Texture Map Dimension *1D***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | *c0* - array dimension in the *t0* direction |
| 2 | | unused |

**Encoding: TEXEL_ARRAY_COUNTS for Texture Map Dimension *2D***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | *c0* - array dimension in the *t0* direction |
| 2 | CARD16 | *c1* - array dimension in the *t1* direction |

**Encoding: TEXEL_ARRAY_COUNTS for Texture Map Dimension *3D***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | *c0* - array dimension in the *t0* direction |
| 2 | CARD16 | *c1* - array dimension in the *t1* direction |
| 2 | CARD16 | *c2* - array dimension in the *t2* direction |
| 2 | | unused |

### 14.1.1.3 TEXEL

**Encoding: TEXEL for Texel Type *LuminanceFloat***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | luminance |

**Encoding: TEXEL for Texel Type *LuminanceInt8***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | luminance |

**Encoding: TEXEL for Texel Type *LuminanceInt16***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *2* | CARD16 | luminance |

**Encoding: TEXEL for Texel Type *LuminanceAlphaFloat***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | luminance |
| *fp* | FLOAT | alpha |

**Encoding: TEXEL for Texel Type *LuminanceAlphaInt8***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | luminance |
| 1 | CARD8 | alpha |

**Encoding: TEXEL for Texel Type *LuminanceAlphaInt16***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | luminance |
| 2 | CARD16 | alpha |

**Encoding: TEXEL for Texel Type *RGBFloat***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | red |
| *fp* | FLOAT | green |
| *fp* | FLOAT | blue |

**Encoding: TEXEL for Texel Type *RGBInt8***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | red |
| 1 | CARD8 | green |
| 1 | CARD8 | blue |

**Encoding: TEXEL for Texel Type *RGBInt16***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | red |
| 2 | CARD16 | green |
| 2 | CARD16 | blue |

**Encoding: TEXEL for Texel Type *RGBAlphaFloat***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| *fp* | FLOAT | red |
| *fp* | FLOAT | green |
| *fp* | FLOAT | blue |
| *fp* | FLOAT | alpha |

**Encoding: TEXEL for Texel Type *RGBAlphaInt8***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | red |
| 1 | CARD8 | green |
| 1 | CARD8 | blue |
| 1 | CARD8 | alpha |

**Encoding: TEXEL for Texel Type *RGBAlphaInt16***

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | CARD16 | red |
| 2 | CARD16 | green |
| 2 | CARD16 | blue |
| 2 | CARD16 | alpha |

### 14.1.2 Create Mipmap Color TM From Resources

**Name**

**PEXCreateColorMipmapTMFromResources**

**Request**

| | |
|---|---|
| *fp_format:* | FLOAT_FORMAT |
| *texture_map*: | TEXTURE_MAP_ID |
| *dimension*: | ENUM_TYPE_INDEX |
| *num_levels*: | CARD16 |
| *texel_type*: | ENUM_TYPE_INDEX |
| *drawable_id*: | DRAWABLE_ID |
| *luminance_channel*: | CARD16 |
| *alpha_channel*: | CARD16 |
| *counts*: | LISTofTEXEL_ARRAY_COUNTS |
| *color_data*: | LISTofRESOURCE_ID |
| *alpha_data*: | LISTofRESOURCE_ID |

**Errors**

Alloc
The server failed to allocate a texture map resource.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Drawable
*drawable_id* is invalid.

IDChoice
ID already in use or not in range assigned to client.

Match
1) A specified resource does not contain luminance data appropriate to the texel type and channel.
2) A specified resource does not contain alpha data appropriate to the texel type and channel.
3) A specified resource is of an unsupported type.
4) A pixmap is not a valid example drawable for this request.

Value
1) The texture map dimension given is not valid.
2) The texture map texel type given is not valid.
3) The number of resources given does not match the number of resources expected.
4) The texture map size is required to be a power of two.
5) The texture map size is required to be square.
6) Luminance or alpha channel is invalid.

**Description**

This request creates a mipmap (§ 14.1.1 ) texture resource for use in modifying the intrinsic color of rendered primitives. *dimension* indicates the dimension (Table 14-2 ) of the texture map. The list of color and alpha resource IDs provides the actual texel data. The visual of the drawable specified by *drawable_id* indicates the source texel format. *texel_type* indicates the texel components of the newly created texture resource. If the *texel_type* (Table 14-3 ) indicates luminance is included, *luminance_channel* (Table 14-4 ) indicates whether one of three channels of RGB are given as luminance data, or if all three channels are combined into a single luminance value. If the *texel_type* indicates luminance is not used, *luminance_channel* is ignored. If the *texel_type* indicates alpha is included, *alpha_channel* (Table 14-4 ) indicates whether one of three channels of RGB are given as alpha data, or if all three channels are combined into a single alpha value. If the *texel_type* indicates that alpha is not included, *alpha_channel* and *alpha_data* are ignored.

**Table 14-4 . *Channel Selection* Constant Values**†1

| Value | Symbol | Description |
|---|---|---|
| 0 | *TMChannelRed* | Select the red channel, channel 0. |
| 1 | *TMChannelGreen* | Select the green channel, channel 1. |
| 2 | *TMChannelBlue* | Select the blue channel, channel 2. |
| 8 | *TMChannelNone* | Used when no luminance or alpha channel is needed. |
| 16 | *TMChannelNTSC* | Indicates the three channels of red, green and blue are to be combined into a single luminance or alpha value using the NTSC encoded values as follows: 0.299*red + 0.587*green + 0.114*blue |

†1) Where there is overlap, the entries of this table match the entries of Table 7-12 (§7.2.17.3), which defines the channel selectors for texture mapping composition method *Decal*.

Texture coordinates are normalized so that, in each dimension, the first through last texel correspond to the texture coordinate range 0.0 to 1.0.

The *counts* parameter provides an array of texture sizes, one size, *c0*, for each level. The required texels are taken starting at the lower-left corner of the resource (the last row of the resource), and proceeding along in the *x* direction. If the resource width is less than the texture size *c0*, then successively higher (earlier) rows are used following the bottom row.

In the two-dimensional texture map case, again one color resource, and optionally one alpha resource, defines each level of the texture map, starting with the base level. Texture map coordinate t0 = t1 = 0.0 corresponds to the lower-left of the image defined by the source. That is, the bottom (last) row of the source supplies texels for the bottom (first) row of the texture. The *x* direction of the resource corresponds to the t0 direction of the texture map. The texture map dimensions for a given level cannot exceed the corresponding resource dimensions.

In the three-dimension texture map case, *c2* color resources, and optionally *c2* alpha resources, define each level of the texture map. *c2,* defined by parameter *counts* for each level, is the third dimension of the level. The *c2* resources defining each level are ordered from t2 = 0.0 to t2 = 1.0. All *c2* resources defining the base level are given first, followed by all the resources defining the next level, and so on.

Inquire *MaxFastTMSize* with request **PEXGetImpDepConstants** to determine whether there is a maximum size for any dimension on the base level which must be respected to achieve optimized implementation. If PEX has such a limit, it treats larger texture maps in an implementation-dependent manner. PEX implementations may have other restrictions on the dimensions of each level of the texture map. Inquire *PowerOfTwoTMRequired* with **PEXGetImpDepConstants** to determine if each dimension must be a power of two. Inquire *SquareTMRequired* with the same request to determine if all the dimensions on each level must be equal to one another.

*drawable_id* indicates the type (visual) of resources. Both the *color_data* and *alpha_data* resources must be of the same type. The colormap associated with the drawable is used to determine the color values.

Not all resource types are supported for creating texture map resources. Table 14-5 lists the values that can be inquired via **PEXGetEnumeratedTypeInfo** to determine which resource types are supported.

**Table 14-5 . Enumerated Type *SupportedTexelResources***

| Value | Mnemonic | Description |
|---|---|---|
| 0 | *Window* | The supplied resource is a mapped window. Treatment of obscured portions of the window is implementation-dependent if there is no backing store for the window. Consistent results require a completely unobscured window. |
| 1 | *Pixmap* | The supplied resource is a pixmap. |
| 2 | *Buffer* | The supplied resource is a MBX buffer. If the buffer is currently displayed, treatment of obscured portions is implementation dependent if there is no backing store for the buffer. Consistent results require a completely unobscured buffer. |
| 3 | *Photomap* | The supplied resource is an XIE photomap. |

Floating point values will be in the floating point format specified in *fp_format*.

Once this function returns, all texel data source resources may be reused for other purposes.

**Encoding: PEXCreateColorMipmapTMFromResources**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 125 | PEX opcode |
| 2 | 6+*m***d*/4+*n**(*a*+1) | request length |
| 2 | FLOAT_FORMAT | floating point format |
| 2 | ENUM_TYPE_INDEX | texture map dimension (see Table 14-2 (§14.1.1)) |
| 4 | TEXTURE_MAP_ID | texture map ID |
| 2 | CARD16 | number of levels (*m*) |
| 2 | ENUM_TYPE_INDEX | texel type (see Table 14-3 (§14.1.1)) |
| 4 | DRAWABLE_ID | drawable ID |
| 2 | CARD16 | luminance channel selector (see Table 14-4 ) |
| 2 | CARD16 | alpha channel selector (see Table 14-4 ) |
| *m***d* | LISTofTEXEL_ARRAY_COUNTS | array of dimensions for each level (see §14.1.1.2) |
| $n^{\dagger 1}$*4 | LISTofRESOURCE_ID | color resource IDs |
| $a*n^{\dagger 1}$*4 | LISTofRESOURCE_ID | alpha resource IDs<br>*a* = (*texel_type* == (*LumimanceAlphaFloat* \|<br>\| *LuminanceAlphaInt8* \| *LuminanceAlphaInt16*<br>\| *RGBAlphaFloat* \| *RGBAlphaInt8*<br>\| *RGBAlphaInt16*) ? 1 : 0 |

†1) If *dimension* == *3D,* then *n* = the summation of *c2* for each level. Otherwise *c2* = 0, and *n* = *m*.

### 14.1.3 Free Texture Map

**Name**

**PEXFreeTextureMap**

**Request**

*texture_map*: TEXTURE_MAP_ID

**Errors**

Value
*texture_map* is not a valid texture map ID.

**Description**

This request deletes the association between the resource ID and the texture map. The texture map storage will be freed when no other resource references it.

**Encoding: PEXFreeTextureMap**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 126 | PEX opcode |
| 2 | 2 | request length |
| 4 | TEXTURE_MAP_ID | texture map ID |

# 15 Accumulation Buffers

## 15.1 General

The accumulation buffer is an ancillary buffer useful in advanced rendering techniques that generally require rendering the entire scene, or some portion of it, multiple times and "accumulating" the resulting images. Such techniques include, but need not be limited to: antialiasing, motion blur, depth of field, and transparency. For more information on techniques which take advantage of the accumulation buffer, see the paper, "The Accumulation Buffer: Hardware Support for High-Quality Rendering", Computer Graphics, Volume 24, Number 4, August, 1990, pp. 309-318, written by Paul Haeberli and Kurt Akeley.

Accumulation is performed by adding the contents of an image (and possibly ancillary components of an image such as alpha values) to the buffer. The images being added may have a weighting applied to them at the time of accumulation, the weighting of the contents of the accumulation buffer may be adjusted during accumulation, the resulting accumulation of all the images can be averaged at the time the accumulation buffer's contents are returned, or some combination of these techniques may be chosen. At the end of accumulation, the buffer is typically transferred to a drawable to display the resulting image.

Accumulation buffers are not drawables, and cannot be rendered to, rather they contain colors which are the result of the application of the color approximation steps described in (§7.2.3). Therefore accumulation is defined only for component visuals, i.e. those visuals whose X11 pixel values (which are the pixels resulting from PEX rendering) directly specify each of the red, green and blue components. See the definition of *DirectColor* and *TrueColor* visuals in the core X11 specification for more detail on visual types.

Since accumulation is performed on images, the accumulation buffer is originally loaded from a drawable and has drawables accumulated into it. Use request **PEXGetAncillaryBufferInfo** (§4.1.1) to determine whether a drawable supports an accumulation buffer, and, if so, the characteristics of that accumulation buffer. If the server implementation supports drawables with an alpha buffer capability, a PEX implementation supporting accumulation buffers must support them with sufficient alpha planes to allow for accumulation of the ancillary alpha values. Additionally the total depth of the accumulation buffer must be equal to or greater than the total depth of the drawable to which it is associated, and the depth of each component (red, green, blue, and if supported, alpha) must also be equal to or greater than the depth of each component. Also, since the accumulation buffer is an ancillary buffer to the drawable, it will have the same width and height as the drawable.

Typical methods of using the accumulation buffer are as follows:

1) Accumulate source images with source weights between 0.0 and 1.0, a destination weight of 1.0, and return without scaling. One application of this approach is for the blending of images into an accumulation buffer whose depth is not sufficiently deeper than the source image to allow for overflow, also known as pixel saturation, on addition. The final image returned will be the non-normalized result of the multiple accumulations.

2) Accumulate source images with source and destination weights of 1.0, and return after scaling by $1/n$, where $n$ is the number of accumulations. For sufficiently deep accumulation buffers where overflow is generally not a problem, the image is added to the accumulation buffer. When the image is returned it will be averaged by the number of accumulations. Some implementations may be able to recognize the 1.0 source weight and skip the multiplication on each accumulation of each pixel.

3) If the accumulation buffer is not sufficiently deep to avoid the possibility of overflow, and the number of accumulations is not known until all the accumulations are complete, it is possible to adjust the running average by using both the source and destination weight factors. The return is then done without scaling.

## 15.2 Response to Drawable Changes

If the drawable with which an accumulation buffer is associated is resized, all subsequent **PEXAccumulateBuffer** and **PEXReturnAccumulationBuffer** requests are ignored until a **PEXLoadAccumulationBuffer** or **PEXFreeAccumulationBuffer** request is made. The ignored accumulation buffer requests cause no action, including the generation of errors. However a **PEXLoadAccumulationBuffer** may generate an *Alloc* error, if an attempt to reallocate the accumulation buffer, due to its drawable being resized, fails.

If the drawable with which an accumulation buffer is associated is destroyed, an implicit **PEXFreeAccumulationBuffer** request is performed by the server.

If the drawable is moved, exposed, or occluded, the server continues processing subsequent accumulation buffer requests using the newly modified drawable.

## 15.3 Accumulation Buffer Requests

These requests manage the creation, freeing, loading and return of accumulation buffer contents. They are listed in Table 15-1 .

**Table 15-1 . Accumulation Buffer Requests**

| **Request** | **Purpose** |
|---|---|
| **PEXAccumulateBuffer** | Accumulate a drawable into an accumulation buffer. |
| **PEXAllocateAccumulationBuffer** | Allocate the ancillary accumulation buffer. |
| **PEXFreeAccumulationBuffer** | Free the ancillary accumulation buffer. |
| **PEXLoadAccumulationBuffer** | Copy a drawable to an accumulation buffer. |
| **PEXReturnAccumulationBuffer** | Copy an accumulation buffer to a drawable. |

### 15.3.1 Accumulate Buffer

**Name**

**PEXAccumulateBuffer**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *drawable_id*: | DRAWABLE_ID |
| *source_weight*: | FLOAT |
| *dest_weight*: | FLOAT |

**Errors**

Drawable
*drawable_id* does not identify a valid drawable.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Match
Drawable identified by *drawable_id* does not have an allocated accumulation buffer.

**Description**

The contents of the drawable ($D$) identified by *drawable_id* are accumulated into the ancillary accumulation buffer ($A$) associated with the drawable using the *source_weight* ($w_s$) and *dest_weight* ($w_d$) as follows:

$$A_{x,y} = A_{x,y} w_d + D_{x,y} w_s$$ **Equation 15.1**

If underflow or overflow occurs, the result is implementation-dependent.

If the drawable identified by *drawable_id* has been resized but its ancillary accumulation buffer has not yet been resized by request **PEXLoadAccumulationBuffer**, this request returns without taking any action.

**Encoding: PEXAccumulateBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 127 | PEX opcode |
| 2 | 3+2**fp* | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | DRAWABLE_ID | drawable resource identifier |
| *fp* | FLOAT | source weight |
| *fp* | FLOAT | destination weight |

### 15.3.2 Allocate Accumulation Buffer

**Name**

**PEXAllocateAccumulationBuffer**

**Request**

*drawable_id*: DRAWABLE_ID

**Errors**

Match

*drawable_id* does not identify a valid drawable or one capable of supporting an accumulation buffer.

Alloc

The server was unable to allocate the necessary resources.

**Description**

An accumulation buffer is allocated for the drawable specified by *drawable_id*. The width and height of the accumulation buffer match that of the drawable. The depth of the buffer is server dependent but must be at least as deep as the drawable (see §15.1). Use request **PEXGetAncillaryBufferInfo** (§4.1.1) to determine the component depths of the accumulation buffer. The contents of the buffer are undefined. Request **PEXLoadAccumulationBuffer** (§15.3.4) is used to initialize the accumulation buffer.

**Encoding: PEXAllocateAccumulationBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 128 | PEX opcode |
| 2 | 2 | request length |
| 4 | DRAWABLE_ID | drawable resource identifier |

### 15.3.3 Free Accumulation Buffer

**Name**

**PEXFreeAccumulationBuffer**

**Request**

| | |
|---|---|
| *drawable_id*: | DRAWABLE_ID |

**Errors**

Drawable
*drawable_id* does not identify a valid drawable.

**Description**

The accumulation buffer associated with *drawable_id* is freed.

If no accumulation buffer is allocated for the drawable, this request returns without further action.

**Encoding: PEXFreeAccumulationBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 129 | PEX opcode |
| 2 | 2 | request length |
| 4 | DRAWABLE_ID | drawable resource identifier |

### 15.3.4 Load Accumulation Buffer

**Name**

**PEXLoadAccumulationBuffer**

**Request**

*drawable_id*: DRAWABLE_ID

**Errors**

Drawable
*drawable_id* does not identify a valid drawable.

Match
Drawable identified by *drawable_id* does not have an allocated accumulation buffer.

Alloc
The server was unable to reallocate an accumulation buffer to match the resized drawable.

**Description**

The content of the drawable indicated by *drawable_id* is copied to the accumulation buffer associated with the drawable, overwriting the existing values in that accumulation buffer.

**Encoding: PEXLoadAccumulationBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 130 | PEX opcode |
| 2 | 2 | request length |
| 4 | DRAWABLE_ID | drawable identifier |

### 15.3.5 Return Accumulation Buffer

**Name**

**PEXReturnAccumulationBuffer**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *drawable_id*: | DRAWABLE_ID |
| *scale_factor*: | FLOAT |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Drawable
*drawable_id* does not identify a valid drawable.

Match
Drawable identified by *drawable_id* does not have an allocated accumulation buffer.

**Description**

The values in the accumulation buffer (*A*) associated with the drawable (*D*) specified by *drawable_id* are multiplied by the *scale_factor* (*w*) and then written to the drawable itself, overwriting the existing values in the drawable. The operation is as follows:

$$D_{x,y} = A_{x,y}w \qquad \textbf{Equation 15.2}$$

If underflow or overflow occurs while scaling, the result is implementation-dependent.

If the drawable identified by *drawable_id* has been resized but its ancillary accumulation buffer has not yet been resized by request **PEXLoadAccumulationBuffer**, this request returns without taking any action.

**Encoding: PEXReturnAccumulationBuffer**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 131 | PEX opcode |
| 2 | 3+*fp* | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | DRAWABLE_ID | drawable resource identifier |
| *fp* | FLOAT | scale factor |

# 16 Search Contexts

## 16.1 General

A search context is a PEX resource that performs a geometric search of a structure network and returns the location and type of the output primitive structure element matching the search criteria. The search context's attributes are described in §16.1.1. A search is initiated by the **PEXSearchNetwork** request (§16.2.8). Because this request operates only on structures, there is no explicit support for immediate mode searching.

Search contexts are X11 resources and carry all the responsibilities and access rights of X11 resources. The requests used to create and manage search context resources are shown in Table 16-1 .

**Table 16-1 . Search Context Resource Management Requests**

| Request | Description |
|---|---|
| **PEXChangeSearchContext** | Modify selected attributes of a search context. |
| **PEXCopySearchContext** | Copy selected attributes from one search context to another. |
| **PEXCreateSearchContext** | Create a search context resource, and initialize selected attributes. |
| **PEXFreeSearchContext** | Free a search context resource. |
| **PEXGetSearchContext** | Retrieve selected attributes from a search context. |
| **PEXSearchNetwork** | Conduct a spatial search in WCs on the structure network associated with a search context. |

### 16.1.1 Search Context Attributes

A search context's attributes hold the criteria used by the context when a search is performed. Only output primitives matching all the search criteria are selected by the search. The search context attributes are given default values when the search context is created if not assigned values in the create request's item list. The values are changed with the **PEXChangeSearchContext** request.

#### 16.1.1.1 Start Path

The *start path* identifies the structure network to search and indicates the element after which to begin searching. The path consists of a list of structure-ID and element-position pairs, each pair but the last giving the location of an Execute Structure, Conditional Execute Structure, Include Structure, or Conditional Include Structure element in the network. A search begins with the element immediately succeeding the element identified by the last element of the *start path*. The default start path is *Null*.

A search path need not reflect an existing path defined in the server's structures until a **PEXSearchNetwork** request is invoked.

#### 16.1.1.2 Search Position and Search Distance

The *search position* and *search distance* form the context's search volume in world coordinate space. Only output primitives that intersect this volume are selected. The search volume, for example a cube or sphere, is centered at the *search position* and has a half-width equal to the *search distance*. If the *search distance* is non-positive, a primitive must intersect the *search position* in order to be selected.

#### 16.1.1.3 Search Ceiling

The *search ceiling* indicates at what point in a structure network a search is to end. It is an index into the *start path*, and indicates the structure at which the search should stop once all that structure's elements have been searched. The default *search ceiling* is 1, which causes the full structure network passed to **PEXSearchNetwork** requests to be searched.

#### 16.1.1.4 Model Clipping Flag

The *model clipping flag* determines whether model clipping is performed during a search. Primitives clipped away by model clipping do not match the search criteria. If the flag is *True*, model clipping is performed according to the model clipping output commands encountered during traversal of a structure network. The initial model clipping indicator (§6.9.59) in this case is *NoClip*, and the model clipping volume (§6.9.60) is *Null*. If the *model clipping flag* is *False*, no model clipping is performed during a search, overriding any output commands that enable model clipping in the searched network.

#### 16.1.1.5 Normal and Inverted Lists

The normal and inverted lists each hold a list of filters. A filter is a combination of an inclusion name set and an exclusion name set. To be selected during a search, an output primitive must be accepted by each filter in the pipeline context's normal list and not accepted by each filter in the context's inverted list. To be accepted by a filter in the normal or inverted list, one of a primitive's names, as determined by Add Names To Set OC (§6.9.1), shall be in the filter's inclusion name set, and the filter's exclusion name set must not contain any of its names. The intention is that filters in the normal list act like the pick filter of the renderer's pipeline state in that it identifies primitives eligible for selection by picking, and that filters in the inverted list act like the pipeline state's invisibility filter, which indicates primitives that are not visible and are thus not normally eligible for selection.

#### 16.1.1.6 Text Font Table

The *text font table* is the resource identifier of the text font table which is used to resolve references by Text Font output commands. When the *text font table* is *Null*, the font assumed by the server is implementation-dependent.

## 16.2 Search Context Requests

### 16.2.1 Item Mask and Item List

Some search context requests have *item_mask* and *item_list* parameters. *item_mask* indicates which values of the search context are to be affected by the request. *item_list* gives those values. The bit assignments for the item mask are given in Table 16-2 . The values must be ordered in *item_list* in the same order as listed in Table 16-2 . Only those values indicated in *item_mask* are to be included in *item_list*. Each value encoding is padded to a multiple of four bytes. Values which could be encoded in less space are assumed to be placed in the least significant bits of the word.

### 16.2.2 Search Context Attribute Encodings

The encodings of the search context attributes are in Table 16-2 . These encodings are used in **PEXCreateSearchContext**, **PEXChangeSearchContext**, and **PEXGetSearchContext** requests.

**Table 16-2 . Search Context Attribute Encodings**

| #of Bytes | Data Type or Value | Default Value | Item Bit |
|---|---|---|---|
| **search position (§16.1.1.2)** | | | |
| 3**fp* | COORD_3D | (0.0, 0.0, 0.0) 5.0 | 0x00000001 |
| **search distance (§16.1.1.2)** | | | |
| *fp* | FLOAT | 0.0 5.0 | 0x00000002 |
| **search ceiling (§16.1.1.3)** | | | |
| 4 | CARD16 | 1 5.0 | 0x00000004 |
| **model clipping flag (§16.1.1.4)** | | | |
| 4 | BOOLEAN | *False* 5.0 | 0x00000008 |
| **start path (§16.1.1.1)** | | | |
| 4 | CARD32 (*n*) | 0 5.0 | 0x00000010 |
| 8**n* | LISTofELEMENT_REF | | |
| **normal list (§16.1.1.5)** | | | |
| 4 | CARD32 (*n*) | 0 5.0 | 0x00000020 |
| 8**n* | LISTofNAME_SET_PAIR (see §16.2.2.1) | | |
| **inverted list (§16.1.1.5)** | | | |
| 4 | CARD32 (*n*) | 0 5.0 | 0x00000040 |
| 8**n* | LISTofNAME_SET_PAIR (see §16.2.2.1) | | |
| **text font table†1 (§16.1.1.6)** | | | |
| 4 | LOOKUP_TABLE_ID | *Null* 5.2 | 0x00000080 |

†1) When modifying the *text font table* search context attribute, it is permissible to set it to a valid text font table LUT resource identifier or **None**, the *Null* resource identifier value defined by X.

#### 16.2.2.1 Type NAME_SET_PAIR

NAME_SET_PAIR : [incl, excl : NAME_SET_ID]

**Encoding: NAME_SET_PAIR**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | NAME_SET_ID | inclusion name set identifier |
| 4 | NAME_SET_ID | exclusion name set identifier |

The inclusion and exclusion name set identifiers must each be either a valid name set resource identifier or **None**, the *Null* resource identifier value defined by X.

### 16.2.3 Change Search Context

5.0

**Name**

**PEXChangeSearchContext**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *sc_id*: | SEARCH_CONTEXT_ID |
| *item_mask*: | BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

LookupTable
The text font table specified in *item_list* is invalid.

NameSet
A name set specified in *item_list* is invalid.

Path
The specified search path contains an invalid element position or structure ID.

SearchContext
*sc_id* is invalid.

Value
An invalid bit in *item_mask* is set or a value in *item_list* is invalid.

**Description**

This request sets the values indicated by *item_mask* in the search context identified by *sc_id*. The *item_list* contains the new values, listed in the order defined in Table 16-2 (§ 16.2.1). Floating point values in *item_list* are in the floating point format specified by *fp_format*.

**Encoding: PEXChangeSearchContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 58 | PEX opcode |
| 2 | 4+*n*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | SEARCH_CONTEXT_ID | search context ID |
| 4 | BITMASK | item mask (§ 16.2.1) |
| *n* | LISTofVALUE | item list (length implicit) |

### 16.2.4 Copy Search Context

5.0

**Name**

**PEXCopySearchContext**

**Request**

| | |
|---|---|
| *src_sc_id*: | SEARCH_CONTEXT_ID |
| *dest_sc_id*: | SEARCH_CONTEXT_ID |
| *item_mask*: | BITMASK |

**Errors**

SearchContext
*src_sc_id* and/or *dest_sc_id* is invalid.

Value
An invalid bit in *item_mask* is set.

**Description**

This requests copies the values indicated by *item_mask* from the search context identified by *src_sc_id* to the search context identified by *dest_sc_id*, replacing those values in the destination context.

**Encoding: PEXCopySearchContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 55 | PEX opcode |
| 2 | 4 | request length |
| 4 | SEARCH_CONTEXT_ID | source search context ID |
| 4 | SEARCH_CONTEXT_ID | destination search context ID |
| 4 | BITMASK | item mask (§ 16.2.1 ) |

### 16.2.5 Create Search Context

5.0

**Name**

**PEXCreateSearchContext**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *sc_id*: | SEARCH_CONTEXT_ID |
| *item_mask*: | BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

Alloc
The server failed to allocate the search context.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

IDChoice
The resource ID given by *sc_id* is already in use or not in the range assigned to the client.

LookupTable
The text font table specified in *item_list* is invalid.

NameSet
A name set specified in *item_list* is invalid.

Path
The specified search path contains an invalid element position or structure ID.

Value
An invalid bit in *item_mask* is set or a value in *item_list* is invalid.

**Description**

This request creates a search context resource for the specified *sc_id*. The *item_mask* indicates those search context values to be set when the search context is created. The *item_list* contains those values, listed in the order defined in Table 16-2 (§ 16.2.1). Floating point values in *item_list* are in the floating point format specified in *fp_format*.

**Encoding: PEXCreateSearchContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 54 | PEX opcode |
| 2 | 4+*n*/4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | SEARCH_CONTEXT_ID | search context ID |
| 4 | BITMASK | item mask (§ 16.2.1) |
| *n* | LISTofVALUE | item list (length implicit) |

### 16.2.6 Free Search Context

5.0

**Name**

**PEXFreeSearchContext**

**Request**

*sc_id*: SEARCH_CONTEXT_ID

**Errors**

SearchContext
*sc_id* is invalid.

**Description**

The search context identified by *sc_id* is deleted and its association with that ID removed.

**Encoding: PEXFreeSearchContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 56 | PEX opcode |
| 2 | 2 | request length |
| 4 | SEARCH_CONTEXT_ID | search context ID |

### 16.2.7 Get Search Context

5.0

**Name**

**PEXGetSearchContext**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *sc_id*: | SEARCH_CONTEXT_ID |
| *item_mask*: | BITMASK |

**Reply**

| | |
|---|---|
| *item_list*: | LISTofVALUE |

**Errors**

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

SearchContext
*sc_id* is invalid.

Value
An invalid bit in *item_mask* is set.

**Description**

This request returns the values indicated by *item_mask* from the search context identified by *sc_id*. The values are returned in *item_list*, in the order in which they’re defined in Table 16-2 (§16.2.1). Floating-point values are returned in the floating-point format specified in *fp_format*. When a name set value of a search context refers to a name set resource that has been freed, the value *AlreadyFreed* (§3.7.7.120) is returned for that name set ID. Likewise when the *text font table* value of a search context refers to a text font table resource that has been freed, the value *AlreadyFreed* is returned or that text font table ID.

**Encoding: PEXGetSearchContext**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 57 | PEX opcode |
| 2 | 4 | request length |
| 2 | FLOAT_FORMAT | floating-point format |
| 2 | | unused |
| 4 | SEARCH_CONTEXT_ID | search context ID |
| 4 | BITMASK | item mask (§16.2.1) |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *n*/4 | reply length |
| 24 | | unused |
| *n* | LISTofVALUE | item list (See §16.2.2 for item encodings.) |

### 16.2.8 Search Network

5.0

**Name**

**PEXSearchNetwork**

**Request**

| | |
|---|---|
| *sc_id*: | SEARCH_CONTEXT_ID |

**Reply**

| | |
|---|---|
| *found_path*: | LISTofELEMENT_REF |

**Errors**

StructurePermission
An invalid position in a locked structure (§ 12.1.1.3) is given in the *start_path*.

Path
The specified search path contains an invalid element position or structure ID.

SearchContext
*sc_id* is invalid.

**Description**

A spatial search in world coordinates is performed on the structure network associated with the search context identified by *sc_id*. The search begins at the element following the one indicated by the *start_path* attribute of *sc_id*. The search is terminated when the end of the structure indicated by *search_ceiling* has been reached. The first element that meets the search criteria is returned in *found_path*. However if the selected primitive is in a locked structure (§ 12.1.1.3), the primitive's element position in the *found_path* is undefined. If no element is found, 0 is returned as the number of elements in *found_path*. When the search ends, the *start_path* attribute of the search context is set to the value returned in *found_path*, unless the selected primitive is in a locked structure. If this case, the *start_path* is set to the element preceding the first structure reference after the selected primitive in the locked structure, if there is a further structure reference, else the *start_path* is set to the reference to the locked structure, unless this causes the *search_ceiling* to be reached, in which case the *start_path* is set to *Null*.

For a structure element to be considered a candidate for the search, the current name set must be accepted by all the name set pairs in the search context's *normal_list*, and must be rejected by all the name set pairs in the *inverted_list*. If the *normal_list* is empty, all possible names are implicitly accepted by the normal list. If the *inverted_list* is empty, all possible name sets are implicitly accepted by the inverted list. When both lists are empty, therefore, all primitives are considered during the search.

When a text or annotation text primitive is encountered during a search operation, only the origin of the text string is used to determine if the search was successful.

**Encoding: PEXSearchNetwork**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 59 | PEX opcode |
| 2 | 2 | request length |
| 4 | SEARCH_CONTEXT_ID | search context ID |
| **Reply:** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 2*$n$ | reply length |
| 4 | | unused |
| 4 | CARD32 | number of element references in found path ($n$) |
| 16 | | unused |
| 8*$n$ | LISTofELEMENT_REF | found path |

# 17 PHIGS Workstations

**<u>WARNING:</u>** *There have been no updates to the PHIGS workstation or pick measure resources in PEX 5.2. The functionality provided is backwards compatible with PEX 5.1. Functionality introduced in PEX 5.2 in areas other than the PHIGS workstation interacts with the PHIGS workstation and pick measure resources in an undefined and implementation-dependent manner. Clients are cautioned not to rely on any functionality other than what was defined in PEX 5.1. The PHIGS workstation and pick measure resources are retained in PEX 5.2 for backwards compatibility only.*

## 17.1 General

A *PHIGS workstation* is a PEX resource that combines other resources into a single entity that behaves in a manner similar to the PHIGS abstraction of a "workstation". The PHIGS workstation has built into it all of the capabilities of the renderer resource described earlier. A PHIGS workstation, with its lookup tables and name sets, effectively contains an instance of a renderer, but it also contains functionality above and beyond that found in a renderer resource.

This section describes the operations that can be performed on PHIGS workstation resources and the operations that can be performed using PHIGS workstation resources. In addition to the attributes listed below, the PHIGS workstation resource also contains an implementation-dependent number of pick device descriptors that support picking operations. These can be set and queried with requests found in §18.

There are several differences between PHIGS workstations and renderers. Due to the desire to match PHIGS semantics for dealing with views, a PHIGS workstation resource has a built-in view table that can only be accessed through the defined PHIGS workstation requests. However, this view table is functionally the same as that used by a renderer, and information about it may be obtained by using the lookup table requests to obtain information about tables of type *View*. A PHIGS workstation resource that has its associated drawable destroyed will be freed implicitly.

Lookup table and name set resources can be bound to a PHIGS workstation resource and then freed. The contents of these resources will remain, since they are still being referenced by the PHIGS workstation. However, when a PHIGS workstation's attributes are queried, the value *AlreadyFreed* will be returned for those resources that have been freed and thus no longer have a valid resource ID by which they can be referenced.

The PHIGS workstation is an X11 resource and carries all of the responsibilities and access rights of X11 resources.

## 17.2 PHIGS Workstation Resource Attributes

The attributes of a PHIGS workstation resource are listed in Table 17-1 . In the table, the abbreviation "create wks req." indicates that the parameter is set by the **PEXCreatePhigsWKS** request. The abbreviation "imp. dep." means that the default value is implementation-dependent.

The encodings of the PHIGS workstation attributes follow. Some PHIGS workstation requests have parameter *item_mask* of type WKS_BITMASK and parameter *item_list* of type LISTofVALUE. *item_mask* indicates which values of the table are to be affected by the request. *item_list* holds those values. The values must be ordered in *item_list* proceeding from the least significant bit of *item_mask* to the most significant bit. Only those values indicated by *item_mask* are to be included in *item_list*. Each value encoding is padded to a multiple of four bytes. Values which could be encoded in less space are assumed to be placed in the least significant bits of the word.

**Table 17-1 . Workstation Resource Attributes**

| Attribute Name | Data Type | Default | Description |
|---|---|---|---|
| *display update* | ENUM_TYPE_INDEX | *VisualizeEach* | (see §17.6.13) 5.0 |
| *visual state* | CARD8 | *Correct* | (see §17.2.1) 5.0 |
| *display surface* | CARD8 | *Empty* | The current status of the PHIGS workstation's display surface. The display surface will be *Empty* if nothing has been rendered on the drawable, *NotEmpty* otherwise. 5.0 |
| *view update* | CARD8 | *NotPending* | *Pending* or *NotPending*, depending on whether there are view modification requests that have yet to be made current. 5.0 |
| *defined views* | LISTofTABLE_INDEX | {0} | The list of view table indices that are defined in the PHIGS workstation's current view table. The list of defined view indices will be returned in the order of view transformation input priority. 5.0 |
| *workstation update* | CARD8 | *NotPending* | *Pending* or *NotPending*, depending on whether there are workstation transformation requests that have yet to be made current. 5.0 |
| *requested NPC subvolume* | NPC_SUBVOLUME | (0.0,0.0,0.0), (1.0,1.0,1.0) | The value for the PHIGS workstation's NPC subvolume specification that has been requested but has not yet been made current. 5.0 |
| *current NPC subvolume* | NPC_SUBVOLUME | (0.0,0.0,0.0), (1.0,1.0,1.0) | The current value for the PHIGS workstation's NPC subvolume specification. 5.0 |
| *requested worksta-tion viewport* | VIEWPORT | {imp. dep., imp. dep., *True*} | The value for the PHIGS workstation's view-port specification that has been requested but has not yet been made current. 5.0 |
| *current worksta-tion viewport* | VIEWPORT | {imp. dep., imp. dep., *True*} | The current value for the PHIGS workstation's viewport specification. 5.0 |
| *HLHSR update* | CARD8 | *NotPending* | *Pending* or *NotPending*, depending on whether there are HLHSR mode requests that have yet to be made current. 5.0 |
| *requested HLHSR mode* | HLHSR_MODE | *Off* | The value for the PHIGS workstation's HLHSR mode that has been requested but has not yet been made current. 5.0 |
| *current HLHSR mode* | HLHSR_MODE | *Off* | The current value for the PHIGS workstation's HLHSR mode. 5.0 |
| *drawable ID* | DRAWABLE_ID | create wks req. | The resource ID of the drawable that is being used as the display surface. 5.0 |
| *marker bundle* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's marker bundle table. 5.0 |

**Table 17-1 . Workstation Resource Attributes (Continued)**

| Attribute Name | Data Type | Default | Description |
|---|---|---|---|
| *text bundle* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's text bundle table. 5.0 |
| *line bundle* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's line bundle table. 5.0 |
| *interior bundle* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's interior bundle table. 5.0 |
| *edge bundle* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's edge bundle table. 5.0 |
| *color table* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the color lookup table that will be used by the PHIGS workstation to dereference indexed color values. 5.0 |
| *depth cue table* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's depth-cue table. 5.0 |
| *light table* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's light table. 5.0 |
| *color approximation table* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the color approximation table that will be used by the PHIGS workstation. 5.0 |
| *pattern table* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's pattern table. 5.0 |
| *text font table* | LOOKUP_TABLE_ID | create wks req. | The resource ID of the PHIGS workstation's text font table. 5.0 |
| *highlight inclusion* | NAME_SET_ID | create wks req. | The resource ID of the PHIGS workstation's highlight inclusion name set. 5.0 |
| *highlight exclusion* | NAME_SET_ID | create wks req. | The resource ID of the PHIGS workstation's highlight exclusion name set. 5.0 |
| *invisibility inclusion* | NAME_SET_ID | create wks req. | The resource ID of the PHIGS workstation's invisibility inclusion name set. 5.0 |
| *invisibility exclusion* | NAME_SET_ID | create wks req. | The resource ID of the PHIGS workstation's invisibility exclusion name set. 5.0 |
| *posted structures* | LISTofSTRUCTURE_INFO | *Null* | The resource ID and associated priority of each of the structures in the PHIGS workstation's posted structure list. The list will be returned in structure priority order. 5.0 |
| *number of priorities* | CARD32 | imp. dep. | The number of display priorities that are supported. A value of zero indicates that a continuous range of priorities is supported. 5.0 |
| *buffer update* | CARD8 | *NotPending* | *Pending* or *NotPending*, depending on whether there are buffer mode requests that have yet to be made current. 5.0 |

**Table 17-1 . Workstation Resource Attributes (Continued)**

| Attribute Name | Data Type | Default | Description |
|---|---|---|---|
| *requested buffer mode* | CARD16 | create wks req. | The value for the PHIGS workstation's buffer mode that has been requested but has not yet been made current (§ 17.5). 5.0 |
| *current buffer mode* | CARD16 | create wks req. | The value for the PHIGS workstation's buffer mode, either *Single* or *Double*. If the drawable associated with the PHIGS workstation resource is a pixmap, then no double-buffering will be performed (§ 17.5). 5.0 |

**Encoding: Workstation Resource Attributes**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| **Display Update** | | |
| 4 | INT16 | |
| **Visual State** | | |
| 4 | CARD8 | |
| | 0 *Correct* | |
| | 1 *Deferred* | |
| | 2 *Simulated* | |
| **Display Surface** | | |
| 4 | CARD8 | |
| | 0 *Empty* | |
| | 1 *NotEmpty* | |
| **View Update** | | |
| 4 | CARD8 | |
| | 0 *NotPending* | |
| | 1 *Pending* | |
| **Defined Views** | | |
| 4 | CARD32 | length (*n*) |
| 4**n* | LISTofCARD16 | |
| **Workstation Update** | | |
| 4 | CARD8 | |
| | 0 *NotPending* | |
| | 1 *Pending* | |
| **Requested NPC Subvolume** | | |
| *fp* | FLOAT | min x |
| *fp* | FLOAT | min y |
| *fp* | FLOAT | min z |
| *fp* | FLOAT | max x |
| *fp* | FLOAT | max y |
| *fp* | FLOAT | max z |

**Encoding: Workstation Resource Attributes (Continued)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| **Current NPC Subvolume** | | |
| *fp* | FLOAT | min x |
| *fp* | FLOAT | min y |
| *fp* | FLOAT | min z |
| *fp* | FLOAT | max x |
| *fp* | FLOAT | max y |
| *fp* | FLOAT | max z |
| **Requested Workstation Viewport** | | |
| 2 | INT16 | min x |
| 2 | INT16 | min y |
| *fp* | FLOAT | min z |
| 2 | INT16 | max x |
| 2 | INT16 | max y |
| *fp* | FLOAT | max z |
| 1 | CARD8 | use drawable |
| | 0 *False* | |
| | 1 *True* | |
| 3 | | unused |
| **Current Workstation Viewport** | | |
| 2 | INT16 | min x |
| 2 | INT16 | min y |
| *fp* | FLOAT | min z |
| 2 | INT16 | max x |
| 2 | INT16 | max y |
| *fp* | FLOAT | max z |
| 1 | CARD8 | use drawable |
| | 0 *False* | |
| | 1 *True* | |
| 3 | | unused |
| **HLHSR Update** | | |
| 4 | CARD8 | |
| | 0 *NotPending* | |
| | 1 *Pending* | |
| **Requested HLHSR Mode** | | |
| 4 | INT16 | |
| **Current HLHSR Mode** | | |
| 4 | INT16 | |
| **Drawable ID** | | |
| 4 | CARD32 | |
| **Marker Bundle Table ID** | | |
| 4 | CARD32 | |

**Encoding: Workstation Resource Attributes (Continued)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| **Text Bundle Table ID** | | |
| 4 | CARD32 | |
| **Line Bundle Table ID** | | |
| 4 | CARD32 | |
| **Interior Bundle Table ID** | | |
| 4 | CARD32 | |
| **Edge Bundle Table ID** | | |
| 4 | CARD32 | |
| **Color Table ID** | | |
| 4 | CARD32 | |
| **Depth Cue Table ID** | | |
| 4 | CARD32 | |
| **Light Table ID** | | |
| 4 | CARD32 | |
| **Color Approximation Table ID** | | |
| 4 | CARD32 | |
| **Pattern Table ID** | | |
| 4 | CARD32 | |
| **Text Font Table ID** | | |
| 4 | CARD32 | |
| **Highlight Inclusion Set ID** | | |
| 4 | CARD32 | |
| **Highlight Exclusion Set ID** | | |
| 4 | CARD32 | |
| **Invisibility Inclusion Set ID** | | |
| 4 | CARD32 | |
| **Invisibility Exclusion Set ID** | | |
| 4 | CARD32 | |
| **Posted Structures** | | |
| 4 | CARD32 | length of list (*n*) |
| (4+*fp*)**n* | LISTofSTRUCTURE_INFO | (§17.2.2) |
| **Number of Priorities** | | |
| 4 | CARD32 | |
| **Buffer Update** | | |
| 4 | CARD8 | |
| | 0 *NotPending* | |
| | 1 *Pending* | |

**Encoding: Workstation Resource Attributes (Continued)**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| **Requested Buffer Mode** | | |
| 4 | CARD16 | |
| | 0 *Single* | |
| | 1 *Double* | |
| **Current Buffer Mode** | | |
| 4 | CARD16 | |
| | 0 *Single* | |
| | 1 *Double* | |

### 17.2.1 Visual State

The PHIGS workstation's current visual state. The visual state will be *Correct* if there are no deferred actions and no changes have been simulated on the display surface. The visual state will be *Deferred* if there are deferred actions, or if there are deferred actions and some changes have been simulated on the display surface. The visual state will be *Simulated* if there are no deferred actions, but some changes have been simulated on the display surface.

### 17.2.2 STRUCTURE_INFO

**Encoding: STRUCTURE_INFO**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 4 | STRUCTURE_ID | structure ID |
| *fp* | FLOAT | priority |

### 17.2.3 WKS_BITMASK

WKS_BITMASK : BITMASK[2]

**Encoding: WKS_BITMASK**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 4 | BITMASK | | PHIGS workstation components |
| | 0x00000001 | *display_update* | |
| | 0x00000002 | *visual_state* | |
| | 0x00000004 | *display_surface* | |
| | 0x00000008 | *view_update* | |
| | 0x00000010 | *defined_views* | |
| | 0x00000020 | *wks_update* | |
| | 0x00000040 | *req_NPC_subvolume* | |
| | 0x00000080 | *cur_NPC_subvolume* | |
| | 0x00000100 | *req_wks_viewpt* | |
| | 0x00000200 | *cur_wks_viewpt* | |
| | 0x00000400 | *HLHSR_update* | |
| | 0x00000800 | *req_HLHSR_mode* | |
| | 0x00001000 | *cur_HLHSR_mode* | |
| | 0x00002000 | *drawable_id* | |
| | 0x00004000 | *marker_bundle* | |
| | 0x00008000 | *text_bundle* | |
| | 0x00010000 | *line_bundle* | |
| | 0x00020000 | *interior_bundle* | |
| | 0x00040000 | *edge_bundle* | |
| | 0x00080000 | *color_table* | |
| | 0x00100000 | *depth_cue_table* | |
| | 0x00200000 | *light_table* | |
| | 0x00400000 | *color_approx_table* | |
| | 0x00800000 | *pattern_table* | |
| | 0x01000000 | *text_font_table* | |
| | 0x02000000 | *highlight_inclusion* | |
| | 0x04000000 | *highlight_exclusion* | |
| | 0x08000000 | *invisibility_inclusion* | |
| | 0x10000000 | *invisibility_exclusion* | |
| | 0x20000000 | *posted_structs* | |
| | 0x40000000 | *num_priorities* | |
| | 0x80000000 | *buffer_update* | |
| 4 | BITMASK | | (second word) |
| | 0x00000001 | *req_buffer_mode* | |
| | 0x00000002 | *cur_buffer_mode* | |

## 17.3 Using Structures with PHIGS Workstations

The further implication of certain structure requests on structures used with the PHIGS workstation resource is specified in this section.

### 17.3.1 Change Structure References

In addition to the semantic described in §12.2.2, the following is also performed. On all PHIGS workstation resources where *new_s_id* is posted, it remains posted and *old_s_id*, if posted there, is unposted. On all PHIGS workstation resources where *new_s_id* is not already posted and *old_s_id* is posted, *new_s_id* is posted with the same priority as *old_s_id* and *old_s_id* is unposted.

### 17.3.2 Destroy Structures

In addition to the semantic described in §12.2.9, the following is also performed. Structures in *list* that are posted to a PHIGS workstation resource are unposted.

## 17.4 Pattern Lookup Table Support

If the PHIGS workstation resource is supported, the server is required to support the pattern table regardless of whether the *Pattern* interior style is supported (§7.2.14).

## 17.5 Double-Buffering Support

Double-buffering with the PHIGS workstation resource requires that the resource be created with the double-buffering mode enabled. To determine if double-buffering can be enabled, the **PEXGetImpDepConstants** request can be used.

## 17.6 PHIGS Workstation Requests

The requests defined in this section manage the creation, get information about, and modify PHIGS workstation resources.

### 17.6.1 Create PHIGS Workstation

5.0

**Name**

**PEXCreatePhigsWKS**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *drawable_id*: | DRAWABLE_ID |
| *marker_bundle*: | LOOKUP_TABLE_ID |
| *text_bundle*: | LOOKUP_TABLE_ID |
| *line_bundle*: | LOOKUP_TABLE_ID |
| *interior_bundle*: | LOOKUP_TABLE_ID |
| *edge_bundle*: | LOOKUP_TABLE_ID |
| *color_table*: | LOOKUP_TABLE_ID |
| *depth_cue_table*: | LOOKUP_TABLE_ID |
| *light_table*: | LOOKUP_TABLE_ID |
| *color_approx_table*: | LOOKUP_TABLE_ID |
| *pattern_table*: | LOOKUP_TABLE_ID |
| *text_font_table*: | LOOKUP_TABLE_ID |
| *highlight_inclusion*: | NAME_SET_ID |
| *highlight_exclusion*: | NAME_SET_ID |
| *invisibility_inclusion*: | NAME_SET_ID |
| *invisibility_exclusion*: | NAME_SET_ID |
| *buffer_mode*: | {*Single*, *Double*} |

**Errors**

IDChoice
ID already in use or not in range assigned to client.

Drawable
The specified drawable resource ID is invalid.

Match
1) One of specified lookup table resources was not created for same root, depth, and visual ID as drawable.
2) The specified drawable is of a type not supported.

LookupTable
One of the specified lookup table resource IDs is invalid.

NameSet
One of the specified name set resource IDs is invalid.

Alloc
1) The server failed to allocate the requested resource.
2) The server cannot allocate resources necessary for double buffering.

Value
The specified *buffer_mode* is invalid.

### Description

This request creates a PHIGS workstation resource for the specified *wks_id*. The window or pixmap specified by *drawable_id* is associated with the newly-created PHIGS workstation resource. The named tables and name sets are also bound to the PHIGS workstation resource for use during rendering. A view table that supports current and requested view table entries is allocated for the PHIGS workstation automatically at creation time. The requests **PEXSetViewRep** and **PEXGetViewRep** can be used to modify and query the PHIGS workstation view table. The *buffer_mode* attribute is used to specify whether the workstation will operate in single-buffered or double-buffered mode. If double-buffered, an additional image buffer will be created for the drawable in an implementation-dependent fashion in order to support double-buffering. Double-buffering is only supported if the value returned by **PEXGetImpDepConstants** for *DoubleBufferingSupported* is *True*. If *drawable_id* is a pixmap, then no double buffering will be performed.

### Encoding: PEXCreatePhigsWKS

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 60 | PEX opcode |
| 2 | 19 | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 4 | CARD32 | drawable ID |
| 4 | CARD32 | marker bundle table ID |
| 4 | CARD32 | text bundle table ID |
| 4 | CARD32 | line bundle table ID |
| 4 | CARD32 | interior bundle table ID |
| 4 | CARD32 | edge bundle table ID |
| 4 | CARD32 | color table ID |
| 4 | CARD32 | depth cue table ID |
| 4 | CARD32 | light table ID |
| 4 | CARD32 | color approximation table ID |
| 4 | CARD32 | pattern table ID |
| 4 | CARD32 | text font table ID |
| 4 | CARD32 | highlight inclusion ID |
| 4 | CARD32 | highlight exclusion ID |
| 4 | CARD32 | invisibility inclusion ID |
| 4 | CARD32 | invisibility exclusion ID |
| 2 | CARD16 | buffer mode |
| | 0 *Single* | |
| | 1 *Double* | |
| 2 | | unused |

### 17.6.2 Execute Deferred Actions

5.0

**Name**

**PEXExecuteDeferredActions**

**Request**

*wks_id*: RESOURCE_ID

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

**Description**

This request causes all the deferred actions on the PHIGS workstation specified by *wks_id* to be executed.

**Encoding: PEXExecuteDeferredActions**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 68 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | PHIGS workstation ID |

### 17.6.3 Free PHIGS Workstation

5.0

**Name**

**PEXFreePhigsWKS**

**Request**

*wks_id*: RESOURCE_ID

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

**Description**

This request deletes the PHIGS workstation resource and any image buffers that were created by the PEX extension in order to support double-buffering. The storage associated with the resource is then freed as well.

**Encoding: PEXFreePhigsWKS**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 61 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | PHIGS workstation ID |

### 17.6.4 Get Dynamics

**Name**

**PEXGetDynamics**

**Request**

| | |
|---|---|
| *drawable_id*: | DRAWABLE_ID |

**Reply**

| | |
|---|---|
| *view_rep*: | DYNAMIC_TYPE (§ 17.6.4.1 ) |
| *marker_bundle*: | DYNAMIC_TYPE |
| *text_bundle*: | DYNAMIC_TYPE |
| *line_bundle*: | DYNAMIC_TYPE |
| *interior_bundle*: | DYNAMIC_TYPE |
| *edge_bundle*: | DYNAMIC_TYPE |
| *color_table*: | DYNAMIC_TYPE |
| *pattern_table*: | DYNAMIC_TYPE |
| *wks_transform*: | DYNAMIC_TYPE |
| *highlight_filter*: | DYNAMIC_TYPE |
| *invisibility_filter*: | DYNAMIC_TYPE |
| *HLHSR_mode*: | DYNAMIC_TYPE |
| *structure_modify*: | DYNAMIC_TYPE |
| *post_structure*: | DYNAMIC_TYPE |
| *unpost_structure*: | DYNAMIC_TYPE |
| *delete_structure*: | DYNAMIC_TYPE |
| *reference_modify*: | DYNAMIC_TYPE |
| *buffer_modify*: | DYNAMIC_TYPE |
| *light_table*: | DYNAMIC_TYPE |
| *depth_cue_table*: | DYNAMIC_TYPE |
| *color_approx_table*: | DYNAMIC_TYPE |

**Errors**

Drawable

The specified drawable resource ID is invalid.

Match

The specified drawable is of a type not supported.

**Description**

This request returns information about the dynamics that are supported by PHIGS workstations associated with drawables of the type specified by *drawable_id*. The dynamics are listed, in order, in Table 17-2 .

**Table 17-2 . PHIGS Workstation Resource Dynamics**

| Attribute Name | Description |
|---|---|
| view rep | Changes to view table |
| marker bundle | Changes to marker bundle |
| text bundle | Changes to text bundle |
| line bundle | Changes to line bundle |
| interior bundle | Changes to interior bundle |
| edge bundle | Changes to edge bundle |
| color table | Changes to color table |
| pattern table | Changes to pattern table |
| wks transform | Changes to workstation transformations |
| highlight filter | Changes to highlight name sets |
| invisibility filter | Changes to invisibility name sets |
| HLHSR mode | Changes to HLHSR mode |
| structure modify | Structure modifications (edits) |
| post structure | Additions to posted structure list |
| unpost structure | Deletions from posted structure list |
| delete structure | Deletion of structure resources |
| reference modify | Structure reference modifications |
| buffer modify | Changes to buffering mode |
| light table | Changes to light table |
| depth cue table | Changes to depth cue table |
| color approx table | Changes to color approximation table |

The value returned for each of the items in the list above can be one of *IMM*, *IRG*, or *CBS*, where *IMM* means that the specified action can be performed and the correct image displayed immediately, *IRG* means that the specified action requires a regeneration of the image, and *CBS* means that the specified action can be simulated immediately if the PHIGS workstation's *display_update* mode is set to *SimulateSome*.

**Encoding: PEXGetDynamics**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 63 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | drawable ID |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 0 | reply length |
| 1 | DYNAMIC_TYPE (§17.6.4.1) | view table entry |
| 1 | DYNAMIC_TYPE | marker bundle |
| 1 | DYNAMIC_TYPE | text bundle |
| 1 | DYNAMIC_TYPE | line bundle |
| 1 | DYNAMIC_TYPE | interior bundle |
| 1 | DYNAMIC_TYPE | edge bundle |
| 1 | DYNAMIC_TYPE | color table entry |
| 1 | DYNAMIC_TYPE | pattern table entry |
| 1 | DYNAMIC_TYPE | workstation transform (window or viewport) |
| 1 | DYNAMIC_TYPE | highlight filter |
| 1 | DYNAMIC_TYPE | invisibility filter |
| 1 | DYNAMIC_TYPE | HLHSR mode |
| 1 | DYNAMIC_TYPE | structure modify |
| 1 | DYNAMIC_TYPE | post structure |
| 1 | DYNAMIC_TYPE | unpost structure |
| 1 | DYNAMIC_TYPE | delete structure |
| 1 | DYNAMIC_TYPE | reference modify |
| 1 | DYNAMIC_TYPE | buffer modify |
| 1 | DYNAMIC_TYPE | light table |
| 1 | DYNAMIC_TYPE | depth cue table |
| 1 | DYNAMIC_TYPE | color approximation table |
| 3 | | unused |

### 17.6.4.1 Type DYNAMIC_TYPE

DYNAMIC_TYPE : {*IMM, IRG, CBS*}

**Encoding: DYNAMIC_TYPE**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | dynamic type |
| | 0 | *IMM* | |
| | 1 | *IRG* | |
| | 2 | *CBS* | |

### 17.6.5 Get View Representation

5.0

**Name**

**PEXGetViewRep**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *index*: | TABLE_INDEX |

**Reply**

| | |
|---|---|
| *view_update*: | {*Pending*, *NotPending*} |
| *requested*: | VIEW_REP |
| *current*: | VIEW_REP |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Value
The specified view table entry is not defined.

**Description**

This request returns the value of the view update state and the specified entries in the requested and current view tables in the PHIGS workstation specified by *wks_id*. The *view_update* will be *Pending* if a view change has been requested but not established. If the specified entry is not defined, an error will be generated and the contents of the reply parameters will be undefined. Floating point values will be returned in the floating point format specified by *fp_format*.

**Encoding: PEXGetViewRep**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 64 | PEX opcode |
| 2 | 3 | request length |
| 2 | INT16 | floating point format |
| 2 | CARD16 | view index |
| 4 | CARD32 | PHIGS workstation ID |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 4+76**fp* | reply length |
| 2 | | view update |
| | 0 *NotPending* | |
| | 1 *Pending* | |
| 22 | | unused |
| 2+38**fp* | VIEW_REP | requested view representation (see §17.6.16.1) |
| 2+38**fp* | VIEW_REP | current view representation (see §17.6.16.1) |

### 17.6.6 Get Workstation Information

5.0

**Name**

**PEXGetWKSInfo**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *item_mask*: | WKS_BITMASK |

**Reply**

| | |
|---|---|
| *item_list*: | LISTofVALUE |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Value
An invalid bit is set in *item_mask*.

**Description**

This request returns attributes of the PHIGS workstation resource indicated by *wks_id*. Floating point values will be returned in the floating point format specified by *fp_format*. The parameter *item_mask* indicates which attributes are to be returned. For each bit that is set in *item_mask*, a corresponding value is returned in *item_list*. The attributes that may be returned, in order, are listed in Table 17-1 (§17.2).

**Encoding: PEXGetWKSInfo**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 62 | PEX opcode |
| 2 | 5 | request length |
| 2 | INT16 | floating-point format |
| 2 | | unused |
| 4 | CARD32 | PHIGS workstation ID |
| 8 | WKS_BITMASK | item mask (see §17.2.3) |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | *n*/4 | reply length |
| 24 | | unused |
| *n* | LISTofVALUE | item list, *n* is size in bytes. See §17.2 encodings. |

### 17.6.7 Get Workstation Postings

5.0

**Name**

**PEXGetWKSPostings**

**Request**

*s_id*: STRUCTURE_ID

**Reply**

*wks_id*: LISTofRESOURCE_ID

**Errors**

Structure
The specified structure resource ID is invalid.

**Description**

This request returns a list of all of the PHIGS workstation resources to which *s_id* has been posted.

**Encoding: PEXGetWKSPostings**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 81 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | structure ID |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | CARD32 | reply length (*n*) |
| 24 | | unused |
| 4**n* | LISTofRESOURCE_ID | PHIGS workstation IDs (length implicit) |

### 17.6.8 Map DC to WC

5.0

**Name**

**PEXMapDCtoWC**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *dc_points*: | LISTofDEVICE_COORD |

**Reply**

| | |
|---|---|
| *wc_points*: | LISTofCOORD_3D |
| *view_index*: | TABLE_INDEX |

**Errors**

PhigsWKS
: The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
: The floating-point format indicated by *fp_format* is not supported.

**Description**

This request maps the device coordinate points in *dc_points* to the world coordinate points in *wc_points* using the PHIGS workstation resource specified by *wks_id*. (The client must convert pointer position values in drawable coordinates into device coordinates.) Each view in the PHIGS workstation's current view table, which has an inverse, is checked to see if it contains all the specified device coordinate points. (If the view transform has no inverse then it is not considered). The index of the view with the highest view transformation input priority that contains all of the points is returned in *view_index*. If no view contains all of the points, the index of the view containing the most points is returned. The points are transformed to world coordinates by passing them through the inverse of the view transform associated with the view index and are returned in *wc_points*. Floating point values are passed and will be returned in the floating point format specified by *fp_format*. Points that are clipped (outside the viewport) will not be transformed and returned in the *wc_points* list, so the number of points returned may be less than the number sent.

**Encoding: PEXMapDCToWC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 71 | PEX opcode |
| 2 | 4+(4+fp)**n*/4 | request length |
| 2 | INT16 | floating point format |
| 2 | | unused |
| 4 | CARD32 | PHIGS workstation ID |
| 4 | CARD32 | number of coordinates (*n*) |
| (4+*fp*)**n* | LISTofDEVICE_COORD | DC points |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | 3**fp***n*/4 | reply length |
| 2 | CARD16 | view index |
| 2 | | unused |
| 4 | CARD32 | number of coordinates returned (*n*) |
| 16 | | unused |
| 3**fp***n* | LISTofCOORD_3D | WC points |

### 17.6.9 Map WC to DC

5.0

**Name**

**PEXMapWCtoDC**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *wc_points*: | LISTofCOORD_3D |
| *view_index*: | TABLE_INDEX |

**Reply**

| | |
|---|---|
| *dc_points*: | LISTofDEVICE_COORD |

**Errors**

PhigsWKS
: The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
: The floating-point format indicated by *fp_format* is not supported.

**Description**

This request maps the world coordinate points in *wc_points* to the device coordinate points in *dc_points* using the PHIGS workstation resource specified by *wks_id* and the view specified by *view_index*. The points are transformed to device coordinates by passing them through the view transform associated with the *view_index*. Floating point values sent and received will be in the floating point format specified by *fp_format*. Points that are clipped will not be returned in the *dc_points* list, so the number of points returned may be less than the number sent.

**Encoding: PEXMapWCToDC**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 72 | PEX opcode |
| 2 | 4+3**fp***n*/4 | request length |
| 2 | INT16 | floating point format |
| 2 | CARD16 | view index |
| 4 | CARD32 | PHIGS workstation ID |
| 4 | CARD32 | number of coordinates (*n*) |
| 3**fp***n* | LISTofCOORD_3D | WC points |
| **Reply** | | |
| 1 | 1 | Reply |
| 1 | | unused |
| 2 | CARD16 | sequence number |
| 4 | (4+*fp*)**n*/4 | reply length |
| 4 | | unused |
| 4 | CARD32 | number of coordinates returned (*n*) |
| 16 | | unused |
| (4+*fp*)**n* | LISTofDEVICE_COORD | DC points |

### 17.6.10 Post Structure

5.0

**Name**

**PEXPostStructure**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *s_id*: | STRUCTURE_ID |
| *priority*: | FLOAT |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Structure
The specified structure resource ID is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

**Description**

This request adds the structure specified by *s_id* to the list of posted structures in the PHIGS workstation *wks_id*. A priority is also provided to indicate the priority of the newly-posted structure with respect to the structures already in the posted structure list. If multiple structures are posted for display to the same display space location, the implementation will ensure the display of the higher priority structure. If two structures have the same priority, the last posted structure has the higher priority. If *s_id* is not a valid structure resource ID, the request is ignored, and an error is generated.

**Encoding: PEXPostStructure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 78 | PEX opcode |
| 2 | 4+*fp*/4 | request length |
| 2 | INT16 | floating point format |
| 2 | | unused |
| 4 | CARD32 | PHIGS workstation ID |
| 4 | CARD32 | structure ID |
| *fp* | FLOAT | priority |

### 17.6.11 Redraw All Structures

5.0

**Name**

**PEXRedrawAllStructures**

**Request**

*wks_id*: RESOURCE_ID

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

**Description**

This request redraws all the posted structures contained in the PHIGS workstation resource specified by *wks_id*. First, if the PHIGS workstation's *display_surface* attribute is *NotEmpty*, its drawable is cleared to the color stored in entry zero of its color lookup table. Then, if any of its *view_update*, *wks_update*, *HLHSR_update*, or *buffer_update* attributes is set to *Pending*, the requested values are made the current values and the attribute is set to *NotPending*. After this, all the posted structures are traversed and rendered (in priority order). Finally, the PHIGS workstation's *visual_state* attribute is set to *Correct*.

**Encoding: PEXRedrawAllStructures**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 65 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | PHIGS workstation ID |

### 17.6.12 Redraw Clip Region

5.0

**Name**

**PEXRedrawClipRegion**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *clip_list*: | LISTofDEVICE_RECT |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

**Description**

This request will perform actions similar to the **PEXRedrawAllStructures** request, except that no PHIGS workstation state attributes are modified or updated by this request. Rendering will occur only in the region defined by *clip_list*. The color stored in entry zero of the *wks_id*'s color lookup table is used to clear the region under the rectangles defined by *clip_list*. The rectangles in *clip_list* should be non-overlapping, or the graphics results will be undefined. (If a Z-buffering algorithm is used, only those pixels under the rectangles in the clip list will have their Z values initialized.) All of the posted structures for *wks_id* are then redrawn, but only pixels under the rectangles in *clip_list* are affected. Pending changes are *not* made current by this request, nor is the *visual_state* attribute modified.

**Encoding: PEXRedrawClipRegion**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 67 | PEX opcode |
| 2 | 3+2**n* | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 4 | CARD32 | number of DC rectangles (*n*) |
| 8**n* | LISTofDEVICE_RECT | clip list |

### 17.6.13 Set Display Update Mode

**Name**

**PEXSetDisplayUpdateMode**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *display_update*: | ENUM_TYPE_INDEX (Table 17-3 ) |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Value
The value given by *display_update* is invalid.

**Description**

This request sets the *display_update* attribute of the PHIGS workstation resource specified by *wks_id*. This attribute defines how changes to the display surface will be visualized. The list of permissible display update modes is defined in Table 17-3 .

If double buffering is enabled (see **PEXSetWKSBufferMode**), the display update mode affects which buffer is rendered into. If the display update mode is *VisualizeNone*, *VisualizeEach* or *VisualizeWhenever*, output primitives are rendered into the back (undisplayed) buffer while the structure network is being traversed. When the traversal is complete, the front and back buffers are swapped, so the rendered image is displayed. If the display update mode is *VisualizeEasy* or *SimulateSome*, output primitives are always rendered into the front (displayed) buffer.

**Table 17-3 . Enumerated Type *DisplayUpdateMode***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *VisualizeEach* | Visualize each change as it occurs. (PHIGS - ASAP) 5.0 |
| 2 | *VisualizeEasy* | Visualize only the changes that are “easy to do”. Things that are “easy to do” are those that have a dynamic modification off IMM or can be updated without a regeneration of the displayed image. The effective result of such an action is equivalent to having performed a regeneration, but without the expense of a complete retraversal and without clearing the display space. (PHIGS - WAIT/UWOR) 5.0 |
| 3 | *VisualizeNone* | Visualize none of the changes. The changes are applied, but the image is not regenerated until there is an explicit request to do so. (PHIGS - WAIT/NIVE) 5.0 |
| 4 | *SimulateSome* | Visualize the easy changes and simulate those changes that can be simulated. (PHIGS - WAIT/UQUM) 5.0 |
| 5 | *VisualizeWhenever* | All changes will eventually be visualized. If regenerations are necessary, they will be performed at the server’s convenience. One regeneration may cause a number of changes to be visualized. The client can issue an update workstation request to guarantee that all changes have been visualized. (PHIGS - ASTI/NIVE) 5.0 |

It should be noted implicit image regenerations may be performed when the display update is one of *VisualizeEach* or *VisualizeWhenever*. If such a regeneration occurs, the display surface will be cleared and any output that was not generated by traversing the posted structure list (such as output from core X) will be lost. *VisualizeEasy*, *VisualizeNone*, and *SimulateSome* will not cause implicit regenerations to occur.

**Encoding: PEXSetDisplayUpdateMode**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 70 | PEX opcode |
| 2 | 3 | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 2 | INT16 | display update mode |
| 2 | | unused |

### 17.6.14 Set HLHSR Mode

5.0

**Name**

**PEXSetHLHSRMode**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *mode*: | HLHSR_MODE |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Value
The value given by *mode* is invalid.

Alloc
The server cannot allocate the necessary resources.

**Description**

This request sets the *req_HLHSR_mode* of the PHIGS workstation specified by *wks_id* to the value in *mode*. If the PHIGS workstation's *display_surface* attribute is *Empty*, or if the dynamic modification for its *HLHSR_mode* is *IMM*, its *cur_HLHSR_mode* attribute is modified with the value contained in *mode* and its *HLHSR_update* is set to *NotPending*; otherwise, its *HLHSR_update* is set to *Pending* and the current value is not changed until the PHIGS workstation is updated.

**Encoding: PEXSetHLHSRMode**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 76 | PEX opcode |
| 2 | 3 | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 2 | INT16 | HLHSR mode |
| 2 | | unused |

### 17.6.15 Set View Priority

5.0

**Name**

**PEXSetViewPriority**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *index1*: | TABLE_INDEX |
| *index2*: | TABLE_INDEX |
| *priority*: | {*Higher*, *Lower*} |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Value
1) The value given by *priority* is invalid.
2) The specified view table entry is not defined.

**Description**

This request sets the relative priorities of entries in *wks_id*'s current view table. The priority of view table entry *index1* with respect to view table entry *index2* is set to the next *Higher* or *Lower* priority. These priorities are used to determine the order in which view table entries are tested when selecting the inverse viewing transformation to use for transformation from device coordinates to world coordinates.

**Encoding: PEXSetViewPriority**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 69 | PEX opcode |
| 2 | 4 | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 2 | CARD16 | index 1 |
| 2 | CARD16 | index 2 |
| 2 | CARD16 | priority |
| | 0 *Higher* | |
| | 1 *Lower* | |
| 2 | | unused |

### 17.6.16 Set View Representation

5.0

**Name**

**PEXSetViewRep**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *view_rep*: | VIEW_REP |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Alloc
The view table is full.

**Description**

This request sets the requested values of the specified view table entry of the view table in the PHIGS workstation specified by *wks_id* to the view representation indicated by *view_rep*. The *view_update* attribute in the PHIGS workstation is set to *Pending* if the change cannot be visualized immediately; otherwise it is set to *NotPending*. If the *view_update* attribute is *NotPending*, the current view table entry is set to the requested values; otherwise the current values are not changed until the PHIGS workstation is updated.

**Encoding: PEXSetViewRep**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 73 | PEX opcode |
| 2 | 5+38**fp*/4 | request length |
| 2 | FLOAT_FORMAT | floating point format |
| 2 | | unused |
| 4 | RESOURCE_ID | **PHIGS** workstation ID |
| 2+38**fp* | VIEW_REP | view representation (see §17.6.16.1) |

#### 17.6.16.1 Type VIEW_REP

VIEW_REP : [index : TABLE_INDEX,
clip_flags : BITMASK_SHORT,
clip_limits : NPC_SUBVOLUME,
orientation, mapping : MATRIX]

**Encoding: VIEW_REP**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 2 | TABLE_INDEX | view table index |
| 2 | | unused |
| 2 | BITMASK_SHORT | requested clip flags |
| | 0x0001 *clip_xy* 5.0 | |
| | 0x0002 *clip_back* 5.0 | |
| | 0x0004 *clip_front* 5.0 | |
| 2 | | unused |
| 6**fp* | NPC_SUBVOLUME | clip limits |
| 16**fp* | MATRIX | view orientation matrix |
| 16**fp* | MATRIX | view mapping matrix |

### 17.6.17 Set Workstation Buffer Mode

5.0

**Name**

**PEXSetWKSBufferMode**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *buffer_mode*: | {*Single*, *Double*} |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Value
The value given by *buffer_mode* is invalid.

Alloc
The server cannot allocate the resources necessary for double buffering (§ 17.5).

**Description**

This request sets the *req_buffer_mode* of the PHIGS workstation specified by *wks_id* to the values in *buffer_mode*. If the PHIGS workstation's *display_surface* attribute is *Empty*, or if the dynamic modification for its *buffer_mode* is *IMM*, its *cur_buffer_mode* attribute is modified with the value contained in *buffer_mode* and its *buffer_update* is set to *NotPending*; otherwise, its *buffer_update* is set to *Pending* and the current value is not changed until the PHIGS workstation is updated.

*Buffer_mode* may be one of the constants *Single* (rendering is to be single-buffered) or *Double* (rendering is to be double-buffered). Double-buffering is only supported if the value returned by **PEXGetImpDepConstants** for *Double-BufferingSupported* is *True*. An *Alloc* error will be returned if the requested mode is *Double* and the server cannot allocate any more image buffers.

**Encoding: PEXSetWKSBufferMode**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 77 | PEX opcode |
| 2 | 3 | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 2 | CARD16 | buffer mode |
| | 0 *Single* | |
| | 1 *Double* | |
| 2 | | unused |

### 17.6.18 Set Workstation Viewport

5.0

**Name**

**PEXSetWKSViewport**

**Request**

*fp_format*: FLOAT_FORMAT
*wks_id*: RESOURCE_ID
*viewport*: VIEWPORT

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Value
The value given by *use_drawable* within the *viewport* parameter is not recognized.

**Description**

This request sets the *req_wks_viewpt* of the PHIGS workstation specified by *wks_id* to the values in *viewport*. The *wks_update* attribute in the PHIGS workstation is set to *Pending* if the change cannot be visualized immediately; otherwise it is set to *NotPending*. If the *wks_update* attribute is *NotPending*, the *cur_wks_viewpt* is set to the requested values; otherwise the current values are not changed until the PHIGS workstation is updated.

**Encoding: PEXSetWKSViewport**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 75 | PEX opcode |
| 2 | 6+2**fp*/4 | request length |
| 2 | INT16 | floating point format |
| 2 | | unused |
| 4 | CARD32 | PHIGS workstation ID |
| 2 | INT16 | viewport min x |
| 2 | INT16 | viewport min y |
| *fp* | FLOAT | viewport min z |
| 2 | INT16 | viewport max x |
| 2 | INT16 | viewport max y |
| *fp* | FLOAT | viewport max z |
| 1 | BOOLEAN | viewport use drawable flag |
| 3 | | unused |

### 17.6.19 Set Workstation Window

5.0

**Name**

**PEXSetWKSWindow**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *NPC_subvolume*: | NPC_SUBVOLUME |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

**Description**

This request sets the *req_NPC_subvolume* of the PHIGS workstation specified by *wks_id* to the values in *NPC_subvolume*. The *wks_update* attribute in the PHIGS workstation is set to *Pending* if the change cannot be visualized immediately; otherwise it is set to *NotPending*. If the *wks_update* attribute is *NotPending*, the *cur_NPC_subvolume* is set to the requested values; otherwise the current values are not changed until the PHIGS workstation is updated.

**Encoding: PEXSetWKSWindow**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 74 | PEX opcode |
| 2 | 3+6**fp*/4 | request length |
| 2 | INT16 | floating point format |
| 2 | | unused |
| 4 | CARD32 | PHIGS workstation ID |
| *fp* | FLOAT | NPC subvolume min x |
| *fp* | FLOAT | NPC subvolume min y |
| *fp* | FLOAT | NPC subvolume min z |
| *fp* | FLOAT | NPC subvolume max x |
| *fp* | FLOAT | NPC subvolume max y |
| *fp* | FLOAT | NPC subvolume max z |

### 17.6.20 Unpost All Structures

5.0

**Name**

**PEXUnpostAllStructures**

**Request**

*wks_id*: RESOURCE_ID

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

**Description**

This request removes all structures from *wks_id*'s posted structure list.

**Encoding: PEXUnpostAllStructures**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 80 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | PHIGS workstation ID |

### 17.6.21 Unpost Structure

5.0

**Name**

**PEXUnpostStructure**

**Request**

| | |
|---|---|
| *wks_id*: | RESOURCE_ID |
| *s_id*: | STRUCTURE_ID |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Structure
The specified structure resource ID is invalid.

**Description**

This request removes the structure specified by *s_id* from *wks_id*'s posted structure list. If *s_id* is not found in the posted structure list, the request is ignored, and an error is generated.

**Encoding: PEXUnpostStructure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 79 | PEX opcode |
| 2 | 3 | request length |
| 4 | CARD32 | PHIGS workstation ID |
| 4 | CARD32 | structure ID |

### 17.6.22 Update Workstation

5.0

**Name**

**PEXUpdateWorkstation**

**Request**

*wks_id*: RESOURCE_ID

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

**Description**

This request will perform actions identical to **PEXRedrawAllStructures** on the PHIGS workstation specified by *wks_id* if its *visual_state* attribute is currently set to *Deferred* or *Simulated*.

**Encoding: PEXUpdateWorkstation**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 66 | PEX opcode |
| 2 | 2 | request length |
| 4 | CARD32 | PHIGS workstation ID |

# 18 PHIGS Workstation Picking

**<u>WARNING:</u>** *There have been no updates to the PHIGS workstation or pick measure resources in PEX 5.2. The functionality provided is backwards compatible with PEX 5.1. Functionality introduced in PEX 5.2 in areas other than the PHIGS workstation interacts with the PHIGS workstation and pick measure resources in an undefined and implementation-dependent manner. Clients are cautioned not to rely on any functionality other than what was defined in PEX 5.1. The PHIGS workstation and pick measure resources are retained in PEX 5.2 for backwards compatibility only.*

## 18.1 General

The discussion of workstation picking includes pick device descriptors which are found in PHIGS workstation resources and pick measures.

## 18.2 Pick Device Descriptors

Each PHIGS workstation resource maintains a list of pick device descriptors. Each entry in this list maintains state values for a particular type of pick device, such as a mouse or a 3D tablet. Together, the entries in the pick device descriptor list maintain state values for all of the pick devices that are supported by the workstation resource. This list of values can be set or inquired for each of the supported pick devices.

Two of the pick device descriptor components are name set resource IDs. If a name set is created, bound to a pick device descriptor, and then freed, the contents of the name set will remain, since it is still being referenced by the pick device descriptor. However, when a pick device descriptor is queried, the value *AlreadyFreed* will be returned for the name set ID, since it no longer has a valid resource ID by which it can be referenced.

The pick device descriptor components, in order, are listed in Table 18-1 . The abbreviation “pick dev. dep.” stands for “pick-device-dependent”, meaning that the default value is determined by the pick device type. The abbreviation “imp. dep.” means that the default value is implementation-dependent.

**Table 18-1 . Pick Device Descriptor Components**

| Attribute Name | Data Type | Default Value | Description |
|---|---|---|---|
| *pick status* | {*NoPick*, *Ok*} | *NoPick* | Contains the initial pick status that will be bound to a pick measure resource. 5.0 |
| *pick path* | LISTofPICK_ELEMENT_REF | *Null* | Contains the initial pick path that will be bound to a pick measure resource. 5.0 |
| *pick path order* | {*TopFirst*, *BottomFirst*} | *TopFirst* | Specifies the order in which elements of the picked path are to be written into a pick measure resource. If the order is *TopFirst*, elements of the pick measure’s *picked_path* attribute will be listed in the order they would have been encountered during a traversal, while if the order is *BottomFirst* they will be listed in the opposite order. 5.0 |
| *pick inclusion* | NAME_SET_ID | *Null* | Specifies the resource ID of the name set resource that is to be used as the pick inclusion filter during picking operations. 5.0 |
| *pick exclusion* | NAME_SET_ID | *Null* | Specifies the resource ID of the name set resource that is to be used as the pick exclusion filter during picking operations. 5.0 |
| *pick data rec* | | pick dev. dep. | Contains a pick-device-dependent data record used to initialize a pick measure resource when it is created. 5.0 |
| *prompt echo type* | ENUM_TYPE_INDEX | pick dev. dep. | Contains an enumerated type value that specifies the prompt/echo type to be used during picking operations. See Table 18-2 for the registered enumerated type values. 5.0 |
| *echo volume* | VIEWPORT | {imp. dep., imp. dep., *True*} | Specifies where prompting and/or echoing is to occur. The default is that the echo volume will be defined to be the size of the drawable that was bound to the PHIGS workstation resource at the time it was created. 5.0 |
| *echo switch* | {*NoEcho*, *Echo*} | *NoEcho* | Specifies the initial echo state for a pick measure. 5.0 |

**Table 18-2 . Enumerated Type *PromptEchoType***

| Value | Mnemonic | Description |
|---|---|---|
| 1 | *EchoPrimitive* | Use an implementation-dependent technique that at least highlights the picked primitive for a short period of time. 5.0 |
| 2 | *EchoStructure* | Echo the contiguous group of primitives with the same pick ID as the picked primitive, or all of the primitives of the structure with the same pick ID as the picked primitive (the extension is free to implement either semantic for this type). 5.0 |
| 3 | *EchoNetwork* | Echo the entire posted structure network that contains the picked primitive. 5.0 |

### 18.2.1 Pick Device Descriptor Component Encodings

The encoding of the pick device descriptor components are given in the following table. All entries in the list are padded to a multiple of 4 bytes. If values appear that could be encoded in less space, it is assumed that they are placed in the least significant bits of the word.

**Encoding: Pick Device Descriptor Components**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| **Pick Status** | | |
| 4 | CARD16 | |
| | 0 *NoPick* | |
| | 1 *Ok* | |
| **Pick Path** | | |
| 4 | CARD32 | number of pick element references (*n*) |
| 12**n* | LISTofPICK_ELEMENT_REF | |
| **Pick Path Order** | | |
| 4 | CARD16 | |
| | 0 *TopFirst* | |
| | 1 *BottomFirst* | |
| **Pick Inclusion** | | |
| 4 | CARD32 | |
| **Pick Exclusion** | | |
| 4 | CARD32 | |
| **Pick Data Record** | | |
| 4 | CARD32 | length of data (*n*) |
| *n* | | |
| *p* | | *p*=pad(*n*) |
| **Prompt and Echo Type** | | |
| 4 | INT16 | |
| **Echo Volume** | | |
| 2 | INT16 | min x |
| 2 | INT16 | min y |
| *fp* | FLOAT | min z |
| 2 | INT16 | max x |
| 2 | INT16 | max y |
| *fp* | FLOAT | max z |
| 1 | BOOLEAN | use drawable |
| 3 | | unused |
| **Echo Switch** | | |
| 4 | CARD16 | |
| | 0 *NoEcho* | |
| | 1 *Echo* | |

## 18.3 Pick Measure

A pick measure resource must be created to actually perform a pick operation. A pick device type (Table 11-3 (§ 11.1.2)) is specified at the time a pick measure resource is created in order to provide the parameters for the picking operation. The pick measure resource accepts input in the form of input records which are defined for each type of pick device. When a pick measure resource is passed a valid input record, its attributes will be updated. Operations on a pick measure resource are potentially lengthy since a great number of structure elements may have to be processed in order to produce the pick results.

The pick measure resource components, in order, are listed in the following table. The abbreviation “from wks” indicates that the default value is inherited from the pick device descriptor stored in the PHIGS workstation resource when the pick measure is created.

**Table 18-3 . Pick Measure Components**

| Attribute Name | Data Type | Default Value | Description |
|---|---|---|---|
| *pick status* | {*NoPick*, *Ok*} | from wks | Contains the result of the last update operation that was performed. It is set to *Ok* if a primitive was successfully picked, and *NoPick* if the pick operation was unsuccessful. 5.0 |
| *picked prim* | LISTofPICK_ELEMENT_REF | from wks | Contains the path of the structure element that was picked as a result of the last update operation. 5.0 |

The pick measure is an X11 resource and carries all the responsibilities and access rights of X11 resources.

### 18.3.1 Pick Measure Component Encodings

The encoding of the pick measure components are given in the following table. All entries in the list are padded to a multiple of 4 bytes. If values appear that could be encoded in less space, it is assumed that they are placed in the least significant bits of the word.

**Encoding: Pick Measure Components**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| **Pick Status** | | |
| 4 | CARD16 | |
| | 0 *NoPick* | |
| | 1 *Ok* | |
| **Picked Primitive** | | |
| 4 | CARD32 | number of pick element references (*n*) |
| 12**n* | LISTofPICK_ELEMENT_REF | |

## 18.4 Using Structures with Pick Measures

The further implication of certain structure requests on structures used with the pick measure resource is specified in this section.

### 18.4.1 Destroy Structures

In addition to the semantic described in §12.2.9, the following is also specified. Paths in pick measures that contain the ID of a structure that is destroyed will not be modified by **PEXDestroyStructures** and may still be queried, but the **PEXUpdatePickMeasure** request generates an error if a path with the ID of a destroyed structure is used.

## 18.5 Workstation Picking Requests

This section defines the requests that initiate and control workstation picking.

### 18.5.1 Change Pick Device Descriptor

5.0

**Name**

**PEXChangePickDevice**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *dev_type*: | PICK_DEVICE_TYPE |
| *item_mask*: | BITMASK |
| *item_list*: | LISTofVALUE |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Value
1) Invalid value for *dev_type*.
2) An item in *item_list* is invalid.
3) An invalid bit is set in *item_mask*.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

Path
The specified pick path contains an invalid structure ID or element offset.

NameSet
The specified name set resource ID is invalid.

**Description**

This request will modify components of a pick descriptor for the PHIGS workstation resource specified by *wks_id*. The descriptor to be modified will be the currently-defined descriptor for the pick device of the type specified by *dev_type*. The *item_mask* specifies which components are to be changed. The specified attributes of the pick device descriptor will be set to the values contained in *item_list*. Floating point values in *item_list* will be in the floating point format specified in *fp_format*.

**Encoding: PEXChangePickDevice**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | opcode (X11 assigned) |
| 1 | 83 | | PEX opcode |
| 2 | 5+*n*/4 | | request length |
| 2 | FLOAT_FORMAT | | floating-point format |
| 2 | | | unused |
| 4 | RESOURCE_ID | | PHIGS workstation ID |
| 2 | PICK_DEVICE_TYPE | | device type |
| 2 | | | unused |
| 4 | BITMASK | | item mask |
| | 0x0001 | pick status | |
| | 0x0002 | pick path | |
| | 0x0004 | pick path order | |
| | 0x0008 | pick inclusion | |
| | 0x0010 | pick exclusion | |
| | 0x0020 | pick data record | |
| | 0x0040 | prompt and echo type | |
| | 0x0080 | echo volume | |
| | 0x0100 | echo switch | |
| *n* | LISTofVALUE | | item list (*n* is the size of the list in bytes) |

### 18.5.2 Create Pick Measure

5.0

**Name**

**PEXCreatePickMeasure**

**Request**

| | |
|---|---|
| *pm_id*: | RESOURCE_ID |
| *wks_id*: | RESOURCE_ID |
| *dev_type*: | PICK_DEVICE_TYPE |

**Errors**

IDChoice
*pm_id* is already in use or not in the ranged assigned to the client.

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Alloc
The server failed to allocate the resource.

Value
Invalid value for *dev_type*.

**Description**

This request creates a pick measure resource of the type specified by *dev_type*. The resource ID of the pick measure is specified by *pm_id*. The newly-created pick measure is initialized with the values contained in the appropriate pick device descriptor stored in the PHIGS workstation resource specified by *wks_id*.

**Encoding: PEXCreatePickMeasure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 84 | PEX opcode |
| 2 | 4 | request length |
| 2 | RESOURCE_ID | PHIGS workstation ID |
| 4 | RESOURCE_ID | pick measure ID |
| 2 | PICK_DEVICE_TYPE | device type |
| 2 | unused | |

### 18.5.3 Free Pick Measure

5.0

**Name**

**PEXFreePickMeasure**

**Request**

*pm_id*: RESOURCE_ID

**Errors**

PickMeasure
*pm_id* is not a valid pick measure resource ID.

**Description**

This request deletes the pick measure resource and frees the storage associated with it.

**Encoding: PEXFreePickMeasure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 85 | PEX opcode |
| 2 | 2 | request length |
| 4 | RESOURCE_ID | pick measure ID |

### 18.5.4 Get Pick Device Descriptor

5.0

**Name**

**PEXGetPickDevice**

**Request**

| | |
|---|---|
| *fp_format*: | FLOAT_FORMAT |
| *wks_id*: | RESOURCE_ID |
| *dev_type*: | PICK_DEVICE_TYPE |
| *item_mask*: | BITMASK |

**Reply**

| | |
|---|---|
| *item_list*: | LISTofVALUE |

**Errors**

PhigsWKS
The specified PHIGS workstation resource ID is invalid.

Value
1) Invalid value for *dev_type*.
2) An invalid bit is set in *item_mask*.

FloatingPointFormat
The floating-point format indicated by *fp_format* is not supported.

**Description**

This request will return components of a pick descriptor for the PHIGS workstation resource specified by *wks_id*. The descriptor returned will be the currently-defined descriptor for the pick device of the type specified by *dev_type*. The *item_mask* specifies which components are to be inquired and returned. The specified attributes of the pick device descriptor will be returned in *item_list*. Floating point values in *item_list* will be returned in the floating point format specified in *fp_format*.

**Encoding: PEXGetPickDevice**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | opcode (X11 assigned) |
| 1 | 82 | | PEX opcode |
| 2 | 4 | | request length |
| 2 | FLOAT_FORMAT | | floating-point format |
| 2 | PICK_DEVICE_TYEP | | device type |
| 4 | RESOURCE_ID | | PHIGS workstation ID |
| 4 | BITMASK | | item mask |
| | 0x0001 | pick status | |
| | 0x0002 | pick path | |
| | 0x0004 | pick path order | |
| | 0x0008 | pick inclusion | |
| | 0x0010 | pick exclusion | |
| | 0x0020 | pick data record | |
| | 0x0040 | prompt and echo type | |
| | 0x0080 | echo volume | |
| | 0x0100 | echo switch | |
| **Reply** | | | |
| 1 | 1 | | Reply |
| 1 | | | unused |
| 2 | CARD16 | | sequence number |
| 4 | *n*/4 | | reply length |
| 24 | | | unused |
| *n* | LISTofVALUE | | item list (*n* is size of list in bytes) |

### 18.5.5 Get Pick Measure Attributes

5.0

**Name**

**PEXGetPickMeasure**

**Request**

*pm_id*: RESOURCE_ID
*item_mask*: BITMASK

**Reply**

*item_list*: LISTofVALUE

**Errors**

PickMeasure
*pm_id* is not a valid pick measure resource ID.

Value
An invalid bit is set in *item_mask*.

**Description**

This request will return components of the pick measure specified by *pm_id*. The *item_mask* specifies which components are to be inquired and returned. The specified attributes of the pick measure will be returned in *item_list*.

**Encoding: PEXGetPickMeasure**

| # of Bytes | Data Type or Value | | Description |
|---|---|---|---|
| 1 | CARD8 | | opcode (X11 assigned) |
| 1 | 86 | | PEX opcode |
| 2 | 3 | | request length |
| 4 | RESOURCE_ID | | pick measure ID |
| 4 | BITMASK | | item mask |
| | 0x0001 | pick status | |
| | 0x0002 | picked primitive | |
| **Reply** | | | |
| 1 | 1 | | Reply |
| 1 | | | unused |
| 2 | CARD16 | | sequence number |
| 4 | *n*/4 | | reply length |
| 24 | | | unused |
| *n* | LISTofVALUE | | item list (*n* is size of list in bytes) |

### 18.5.6 Update Pick Measure

5.0

**Name**

**PEXUpdatePickMeasure**

**Request**

*pm_id*: RESOURCE_ID
*input_record*: data

**Errors**

PickMeasure
*pm_id* is not a valid pick measure resource ID.

Path
The specified pick path contains an invalid structure ID or element offset.

**Description**

This request will update the pick measure specified by *pm_id* using the information found in *input_record*. If this update causes a primitive to be picked, the pick measure's *pick_status* will be set to *Ok* and the *picked_prim* will be set to the path of the picked primitive. If no primitive was picked, the *pick_status* will be set to *NoPick*.

*picked_prim* can be used for echoing when the pick measure is created. However it is not used as a start path from which to start picking.

The data in *input_record* is dependent on the type of pick device for which the pick measure resource was created, that is, the *dev_type* specified to **PEXCreatePickMeasure**. **PEXGetPickMeasure** can be used to return the results of a **PEXUpdatePickMeasure** request.

**Encoding: PEXUpdatePickMeasure**

| # of Bytes | Data Type or Value | Description |
|---|---|---|
| 1 | CARD8 | opcode (X11 assigned) |
| 1 | 87 | PEX opcode |
| 2 | 3+(*n*+*p*)/4 | request length |
| 4 | RESOURCE_ID | pick measure ID |
| 4 | CARD32 | length of list (*n*) |
| *n* | | input record |
| *p* | | *p*=pad(*n*) |

The pick measure input data records for the registered pick device types are:

DC_HitBox : [pick_position : DEVICE_COORD_2D, pick_distance : FLOAT]
NPC_HitVolume : [pick_volume : NPC_SUBVOLUME]

The encodings for these are given in the following table.

**Encoding: Pick Measure Data Records**

| | | |
|---|---|---|
| **Pick Device Type 1: DC Hit Box** | | |
| 2 | INT16 | pick position x |
| 2 | INT16 | pick position y |
| *fp* | FLOAT | pick distance |
| **Pick Device Type 2: NPC Hit Volume** | | |
| *fp* | FLOAT | min x |
| *fp* | FLOAT | min y |
| *fp* | FLOAT | min z |
| *fp* | FLOAT | max x |
| *fp* | FLOAT | max y |
| *fp* | FLOAT | max z |

# 19 Conformance Requirements

## 19.1 General

This chapter specifies the capabilities a PEX 5.2 implementation must support to be a conformant 5.2 implementation.

Not all drawables supported by a device supporting PEX are required to support PEX. However at least one drawable which is a window is required.

All functionality, including lists of constant values, must be supported unless explicitly stated otherwise in this chapter or elsewhere in the specification.

## 19.2 Requests

Unless otherwise explicitly specified, the functionality of all requests in the Immediate Rendering (§4.1.2.1) and Structure Rendering (§4.1.2.2) subsets must be fully supported. Support for all optional functionality can be programmatically determined.

## 19.3 Output Commands

Unless otherwise explicitly specified, all output commands (Chapter 6) must be fully supported.

## 19.4 Enumerated Type Values

All enumerated type classes (§4.1.3) must be inquirable. Table 19-1 defines the values required for each class. In the case of enumerated type values registered as implementation-dependent, **PEXGetEnumeratedTypeInfo** must return a mnemonic string which indicates what option is taken. If that option is also one of the standard values, both values are returned and the same mnemonic string is returned for each.

**Table 19-1 . Enumerated Types Required Values**

| Class | Definition | Required Values |
|---|---|---|
| *AlphaBlendingFunction* | §6.9.2 | *NoBlend* and *SimpleBlend*†1 |
| *ATextStyle* | §6.9.6 | *NotConnected* and *Connected* |
| *ColorApproxModel* | §7.2.3 | *RGB* |
| *ColorApproxType* | §7.2.3 | *ColorSpace* |
| *ColorType* | §3.7.6 | *Indexed* and *RGBFloat*†2 |
| *CurveApproxMethod* | §6.9.25 | 1 (implementation-dependent), *ConstantBetweenKnots, WCS_ChordalSize, NPC_ChordalSize, DC_ChordalSize, WCS_ChordalDev, NPC_ChordalDev*, and *DC_ChordalDev* |
| *DataMappingMethod* | §6.9.27 | *None* and *Color* |
| *DisplayUpdateMode* | §17.6.13 | (no required values) |
| *EchoMethod* | §10.2.13 | *DestructiveColor* |

**Table 19-1 . Enumerated Types Required Values (Continued)**

| Class | Definition | Required Values |
|---|---|---|
| *Escape* | §5.1 | (no required values) |
| *FloatFormat* | §3.7.5 | *IEEE_754_32*†3 |
| *GDP* | §6.7.16 | (no required values) |
| *GDP3* | §6.7.15 | (no required values) |
| *GSE* | §6.9.32 | (no required values) |
| *HatchStyle* | §6.9.85 | *45Degrees* and *135Degrees*†4 |
| *HighlightMethod* | §7.2.8 | *Color* |
| *HLHSRIdentifier* | §6.9.38 | *Disable* and *Enable* |
| *HLHSRMode* | §10.2.20 | *Off, ZBuffer,* and *ZBufferId* |
| *InteriorStyle* | §6.9.84 | *Hollow, Solid, Empty,* and *Texture* |
| *LightType* | §7.2.10 | *Ambient, WCS_Vector, WCS_Point,* and *WCS_Spot* |
| *LineCapStyle* | §6.9.43 | *Butt* |
| *LineJoinStyle* | §6.9.46 | 0 (implementation-dependent) |
| *LineType* | §6.9.48 | *Solid, Dashed, Dotted, DashDot, CenterLine,* and *Phantom* |
| *MarkerGlyphType* | §7.2.13 | *Stroke* |
| *MarkerType* | §6.9.58 | *Dot, Cross, Asterisk, Circle,* and *X* |
| *ModelClipOperator* | §6.9.60 | *Replace* and *Intersection* |
| *ParametricSurfaceCharacteristics* | §6.9.63 | *None*, 2 (implementation-dependent), and *IsoparametricCurves* |
| *PickAllMethod* | §11.1.3.2 | *All* |
| *PickDeviceType* | §11.1.2 | *DCHitBox* and *NPCHitVolume* |
| *PickOneMethod* | §11.1.3.1 | *Last* and *ClosestZ* |
| *PolylineInterpMethod* | §6.9.68 | *None* and *Color* |
| *PrimAntiAliasBlendFunction* | §6.9.69 | 0 (implementation-dependent) |
| *PromptEchoType* | §18.2 | (no required values) |
| *ReflectionModel* | §6.9.88 | *None, Ambient, Diffuse*, and *Specular* |
| *RenderingColorModel* | §6.9.72 | 0 (implementation-dependent) |
| *SupportedTexelResources* | §14.1.2 | *Window* and *Pixmap* |
| *SurfaceApproxMethod* | §6.9.75 | 1 (implementation-dependent), *ConstantBetweenKnots, WCS_ChordalSize, NPC_ChordalSize, DC_ChordalSize, WCS_ChordalDev, NPC_ChordalDev*, and *DC_ChordalDev* |
| *SurfaceEdgeType* | §6.9.82 | *Solid* |
| *SurfaceInterpMethod* | §6.9.86 | *None* and *Color* |
| *TMBoundaryClampMethod* | §7.2.17.4.3 | *Discontinue* and *Color* |
| *TMBoundaryCondition* | §7.2.17.4.2 | *ClampExplicitMethod* |
| *TMCompositionMethod* | §7.2.17.3 | *Replace, Modulate,* and *Decal* |
| *TMDimension* | §14.1.1 | *2D* |
| *TMParameterizationMethod* | §7.2.17.2.1 | *FloatData,* and *ReflectionSphereVRC* |

**Table 19-1 . Enumerated Types Required Values (Continued)**

| Class | Definition | Required Values |
|---|---|---|
| *TMPerspectiveCorrection* | §6.9.96 | *None* and *Pixel* |
| *TMRenderingControl* | §7.2.17.1 | *PostLight* |
| *TMResourceHints* | §6.9.97 | *None, Speed,* and *Space* |
| *TMSampleFrequency* | §6.9.98 | *Pixel* |
| *TMSampleMethod* | §7.2.17.4.1 | *SingleBase* and *WeightedBase* |
| *TMTexelType* | §14.1.1 | *LuminanceFloat, LuminanceInt8, LuminanceAlphaFloat, LuminanceAlphaInt8, RGBFloat, RGBAlphaFloat, RGBInt8,* and *RGBAlphaInt8* |
| *TransparencyMethod* | §10.2.40 | *None* and *AlphaBlend* |
| *TrimCurveApproxMethod* | §6.7.24 | 1 (implementation-dependent) and *ConstantBetweenKnots* |
| *ZTest* | §6.9.101 | *GreaterThanOrEqual* |

†1) The requirement for only alpha blending functions *NoBlend* and *SimpleBlend* imply the alpha buffer is not required. However the alpha buffer must be supported if and only if a server indicates it supports alpha blending functions which utilize destination alpha values.
†2) Enumerated type class *ColorType* value *RGBFloat* implies both homogeneous and non-homogeneous red-green-blue floating-point colors.
†3) When the *FloatFormat* enumerated type class values are inquired, the PEX implementation is required to return its native floating-point format first in the list of values returned.
†4) These hatch styles are required only if interior style *Hatch*, which is optional, is supported. Otherwise no hatch styles are required.

## 19.5 Implementation-Dependent Constants

All implementation-dependent constants (§4.1.4) must be inquirable. Table 19-1 defines the values required for each constant. If there are no required values for a constant, the entry in the **Required Value** column is left blank.

**Table 19-2 . Implementation-Dependent Constants Required Values**

| Constant | Required Value |
|---|---|
| *BestColorApproxValues* | |
| *ChromaticityBlueU* | |
| *ChromaticityBlueV* | |
| *ChromaticityGreenU* | |
| *ChromaticityGreenV* | |
| *ChromaticityRedU* | |
| *ChromaticityRedV* | |
| *ChromaticityWhiteU* | |
| *ChromaticityWhiteV* | |
| *ConcentrationClampSupported* | |
| *DitheringControllable* | |
| *DitheringSupported* | |
| *DoubleBufferingSupported* | |

**Table 19-2 . Implementation-Dependent Constants Required Values (Continued)**

| Constant | Required Value |
|---|---|
| *EyepointUsed* | |
| *LuminanceBlue* | |
| *LuminanceGreen* | |
| *LuminanceRed* | |
| *LuminanceWhite* | |
| *MaxEdgeWidth* | |
| *MaxFastTMSize* | |
| *MaxHitsEventSupported* | *True* |
| *MaxLineWidth* | >= 4 |
| *MaxMarkerSize* | |
| *MaxMarkerVertices* | >= 16 |
| *MaxModelClipPlanes* | >= 6 |
| *MaxNamesetNames* | >= 256 |
| *MaxNonAmbientLights* | >= 7[†1] |
| *MaxNumPickResults* | >= 16 |
| *MaxNURBOrder* | >= 6 |
| *MaxTextureMaps* | >= 1 |
| *MaxTrimCurveOrder* | >= 6 |
| *MinEdgeWidth* | |
| *MinLineWidth* | 1 |
| *MinMarkerSize* | |
| *NominalEdgeWidth* | |
| *NominalLineWidth* | |
| *NominalMarkerSize* | |
| *NumSupportedEdgeWidths* | |
| *NumSupportedLineWidths* | >= 4, or 0 (continuous) |
| *NumSupportedMarkerSizes* | |
| *PixelColorSupported* | |
| *PowerOfTwoTMRequired* | |
| *PrimAntiAliasSupported* | |
| *QuadraticAttenSupported* | |
| *SquareTMRequired* | |
| *TMSamplingHintsUsed* | |
| *TransparencySupported* | |

†1) Additionally, at least 1 simultaneous *Ambient* light is required.

## 19.6 Tables

All types of lookup tables, except pattern, are required. The pattern lookup table is required if interior style *Pattern* is supported or the PHIGS Workstation is supported.

**Table 19-3 . Minimum Table Entries Required**

| Table Type | Minimum Number of Entries Required |
|---|---|
| *Color* | 256 |
| *ColorApprox* | 1 |
| *DataMapping* | 6 |
| *DepthCue* | 2 |
| *EdgeBundle* | 20 |
| *ExtentSize* | 2 |
| *Highlight* | 2 |
| *InteriorBundle* | 20 |
| *Light* | 8 |
| *LineBundle* | 20 |
| *MarkerBundle* | 20 |
| *MarkerGlyph* | 20 |
| *Pattern* | (implementation-dependent) |
| *TextBundle* | 20 |
| *TextFont* | 1 |
| *TextureMapping* | 20 |
| *View* | 256 |

## 19.7 PEX Fonts

At least two fonts must be supported. At least one of them must be mono-spaced. At least one of them must support character set ISO 8859/1 (Latin1) through a stroked PEX font. The XLFD name for this font must match the following XLFD match pattern:

"-*-*-*-*-*-*-*-*-*-*-*-*-ISO8859-1".

All fonts must at least be named with the XLFD naming conventions.

## 19.8 Dependencies on Other Extensions to X

Support for the Double-Buffering Extension to X (DBE) is required if double-buffered rendering is supported. Double-buffering itself is not required, but if double-buffering is supported, it must be at least supported with DBE. Other mechanisms for double-buffering may be supported in addition to DBE for reasons of backwards compatibility.

Large requests and large output commands, where "large" implies the size is too big to be specified in a CARD16 field, can only be supported if the Big Request Extension (BRE) is supported with PEX.

# 20 Caveats for 5.0/5.1 Clients

This chapter documents known caveats for version 5.0/5.1 clients running to a 5.2 server or porting to 5.2.

1. In versions 5.0/5.1, it is not explicitly documented setting undefined bits in the *clip flags* field of a view lookup table results in an error. Clients setting these same bits with a 5.2 server do receive an error.

2. In versions 5.0/51, the Z-buffer is bound to the renderer. This implies the Z-buffer can be shared across multiple drawables. However very few PEX 5.1 servers to date actually support this semantic. Therefore it is unlikely 5.0/5.1 clients take advantage of it.

   In version 5.2, Z-buffers are instead bound to the drawable. The alpha and accumulation ancillary buffers, new with 5.2, are also bound to the drawable. This adds the ability for multiple renderers to use the same ancillary buffer on the same drawable.

3. There have been no updates to the PHIGS workstation or pick measure resources in version 5.2. The functionality provided is backwards compatible with PEX 5.1. Functionality introduced in version 5.2 interacts with the PHIGS workstation and pick measure resources in an undefined and implementation-dependent manner. Clients using these resources are cautioned not to rely on any functionality other than what was defined in version 5.1. The PHIGS workstation and pick measure resources are being retained in version 5.2 for backwards compatibility only.

4. New in version 5.2, the client is explicitly allowed to give **None**, the *Null* resource identifier value defined by X, as a valid resource identifier in some cases. In 5.0/5.1, usage of **None** in these cases may have resulted in an error.