1. Acknowledgements

On behalf of the M.I.T. X Consortium, we would like to thank all who have participated in the design, definition, and implementation of PEX.

This work began when, at the first X3D meeting at MIT in June, 1987, Digital Equipment Corporation and Sun Microsystems, Inc. jointly proposed that a protocol be defined based upon an initial working draft developed by Digital. This draft was influenced by comments from professional colleagues. We recognize especially the authors of that first document: Randi J. Rost, Jeffrey Friedberg, Jeffrey S. Saltz, Pete Nishimoto, and William H. Clifford, Jr., all of Digital Equipment Corporation. They provided a head start for this major undertaking.

We owe a major debt to the PEX architecture team which met frequently between June, 1987 and August, 1991 to argue and debate issues. Members of the PEX Architecture Team: Jeffrey Friedberg, John McConnell, Pete Nishimoto, Randi J. Rost, and Jeffrey S. Saltz of Digital Equipment Corporation; Dave Gorgen, Tom Gross, and Jeff Stevenson of Hewlett-Packard Company; David Plunkett of Solbourne Computer; Jan Hardenbergh of Stardent Computer, Inc.; and Marty Hess, and Jim Van Loo of Sun Microsystems.

Our sincerest thanks are also extended to Bertram Herzog of the University of Michigan, who served as chairman of the X3D organization and was responsible for overseeing a fair and timely process designed to bring the PEX project to fruition.

Several members of the X3D group provided valuable comments to the PEX architecture team. We note especially the contributions of James Michener of Hewlett-Packard Company, and Todd Newman and Raymond Drewry of Digital Equipment Corporation who attended architecture team meetings and provided much-needed insight at critical points during the evolution of PEX. Others who have participated actively in technical discussions include Sally C. Barry, Andy Vesper, and Ray Shapiro of Digital Equipment Corporation.

We also give thanks to early implementors of PEX, including the PEX Sample Implementation team at Sun Microsystems, for providing useful review and sanity checks of the PEX specification documents. The PEX-SI team at Sun includes Marty Hess, Lisa Chabot, Cheryl Huntington, Erwin Hom, Nash Aragam, Tom Gaskins, John Recker, and Chris Nicholas.

Robert Scheifler, Director of the X Consortium at M.I.T., provided invaluable critique and advice that ensured that our efforts conformed to the spirit and definition of the X Window System. Jim Fulton of the X Consortium also contributed to meetings and to technical discussions.

To Randi Rost, the first PEX Document Editor, go the thanks of all, for translating all the decisions into the written word. Sally Barry and Jeff Friedberg are gratefully acknowledged for providing thorough and painstaking review of many intermediate drafts of these early PEX documents. Sally Barry, document editor for PEX 5.0, gratefully acknowledges the invaluable assistance and encouragement of Jeff Friedberg, Jan Hardenbergh, and Randi Rost, during the production of the PEX 5.0 specifications. Paula Womack, document editor for PEX 5.1, acknowledges Ken Garnett of Shographics, for his thorough and timely review of the protocol specification.

In addition the careful review by the PEX teams at all the various companies that are working on PEX implementations has greatly helped the accuracy and precision of this document.

Thanks also to the individuals that worked on the X11, PHIGS, and PHIGS-PLUS specification documents, from which some PEX descriptions were obtained.

Finally, we acknowledge the contributions of all of the other participants in this effort that have not been explicitly named. The contributions of so many individuals has helped to ensure that PEX will be a useful and long-lived extension to the X Window System.

2. PEX Protocol Specification

2.1. Request Processing

Workstation regeneration and requests which process output commands are not guaranteed to be atomic with respect to concurrent rendering to the same destination drawable. However atomicity of execution is guaranteed for each individual output command primitive, excluding Execute Structure, GSE, GDP 3D, and GDP 2D. Also, requests and output commands will be executed in sequential order for a given connection.

2.2. Protocol Format

All PEX† protocol formats are based on the formats specified by the *X Window System Protocol, Version 11*‡, which is referred to as X11 throughout the remainder of this document. The PEX protocol will adhere to the philosophy and requirements of X11.

2.2.1. Request Format

Every X11 request contains a 4-byte header which 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, the length of which is specified by the length field. Since PEX is a proper extension of X11, the 8-bit major opcode contains the opcode assigned for the PEX extension by X11. The length field contains the length of the request in units of four bytes, including the header. The data (or minor opcode) field contains the PEX opcode for this request.

2.2.2. Reply Format

Every reply consists of at least 32 bytes. A header is contained in these 32 bytes. The header of a reply consists of a 32-bit length field, a 16-bit sequence number field, and an 8-bit type field. Zero or more additional bytes follow the header as specified in the length field. The length field specifies the length of the data following the 32 byte reply header and is in units of four bytes. Unused bytes within a reply are not guaranteed to be zero. 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.

2.2.3. Error Format

Error reports are 32 bytes long. Every error includes an 8-bit error code. This error code is used to signify the specific PEX error that occurred. Every error reply also includes the major opcode (the extension reporting the error is identified by the major opcode), the minor opcode (the extension opcode which caused the error), and the least significant 16 bits of the sequence number of the request which had failed. Also included is an 8-bit type field which designates the packet as being an error packet. Unused bytes within an error are not guaranteed to be zero.

2.2.4. Event Format

Events are 32 bytes long. Every event contains an 8-bit type code. The most significant bit in this field is set if the event was generated from a **SendEvent** request. Event codes 64-127 are reserved for extensions. The core X11 protocol does not define a mechanism for expressing interest in events generated by extensions.

2.3. Syntax

Curly braces {...} enclose a set of alternatives. Square brackets [...] enclose a list of structure 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, registered 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 : type1
argN : typeN

Reply:

result1 : type1
resultM : typeM

Errors:

kind1,..., kindK

Description:

Functional description goes here

If no reply description is given, then the request has no reply (it is asynchronous), but errors may still be reported.

2.4. Naming Conventions

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 resources
Copy	Copy attributes from one resource to another of the same type
Get	Return resource attributes from the server to the client
Change	Modify attributes of a resource
Set	Modify a selected attribute of a resource
Destroy	Remove an instance of a resource and all references to it, and deallocate the 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 step of some kind
End	Perform a termination step of some kind

2.5. Common Types

The types listed in this section define the common types used in the PEX protocol specification.

2.5.1. LISTofFOO

A type name of the form LISTofFOO means a counted list of elements of type FOO; the size of the length field may vary (it is not necessarily the same size as FOO). In cases where the number of items in the list is easily computed, the number of items may not be supplied. In all other cases in the PEX protocol (except for LISTofVALUE), the length field is explicit.

2.5.2. BITMASK and LISTofVALUE

The types BITMASK and LISTofVALUE are somewhat special. Various requests contain arguments of the form:

> *item_mask* : BITMASK
> *item_list* : LISTofVALUE

used to allow the client to specify a subset of a heterogeneous collection of "arguments". The *item_mask* specifies which arguments are to be provided; each such argument is assigned a unique bit position. The representation of BITMASK may contain more bits than there are defined arguments; unused bits in the *item_mask* must be zero (or the extension will generate a *Value* error). The *item_list* contains one item for each one bit in the mask, from least to most significant bit in the mask.

2.5.3. Floating Point Format - FLOAT

The PEX protocol allows floating-point values to be passed in various floating-point formats. All floating-point arguments will be specified as FLOAT, which is defined to be the floating-point type contained in the format word associated with the request. Furthermore, items such as MATRIX, VECTOR, and COORD will be in the floating-point format specified by the format word associated with the request.

2.5.4. Colors

In PEX, colors are typically passed as a color type and a value. The color type specifies whether the color is an index value or a direct color value of some type. PEX servers are required to be able to deal with indexed colors and at least one type of direct color. Indexed colors are specified using an index which is used to obtain the color from a color lookup table. Direct colors are specified directly as RGB, HSV, HLS, or CIELUV color values of some form. The list of registered direct color formats can be found in the "Extension Information" section. PEX servers are free to store direct color values in whatever implementation-dependent format they choose, but they must be capable of converting those values back into the originally-specified color type when queried by the client.

2.5.5. Element Types

Chapter 3 describes the set of output commands that are recognized by a PEX implementation. These output commands are distinguished by a 16-bit ELEMENT_TYPE value. This value contains a 16-bit unsigned short that defines the actual type of output command. The high-order bit of the element type is used to signify whether the output command is a standard PEX output command (high-order bit equals zero) or whether the output command is a proprietary addition to the set of standard PEX output commands. Servers are expected to be able to create structure elements containing non-standard PEX output commands, but the execution of such output commands can be a no-op. Since the contents of these output commands is unknown, no floating-point conversions or byte-swapping will be performed on non-standard output commands that are not supported by the server. Unlike the use of the PHIGS-style GSE and GDP output commands, this extension mechanism allows vendors to gracefully add fully-integrated functionality to the standard PEX extension, and permits an implementation to ignore output commands with which it is not familiar.

2.5.6. Types

The PEX Protocol types are as follows:

ASF_ATTRIBUTE	: {*MarkerTypeASF, MarkerScaleASF, MarkerColorASF, TextFontIndexASF, TextPrecASF, CharExpansionASF, CharSpacingASF, TextColorASF, LineTypeASF, LineWidthASF, LineColorASF, CurveApproxASF, PolylineInterpASF, InteriorStyleASF, InteriorStyleIndexASF, SurfaceColorASF, SurfaceInterpASF, ReflectionModelASF, ReflectionAttrASF, BFInteriorStyleASF, BFInteriorStyleIndexASF, BFSurfaceColorASF, BFSurfaceInterpASF, BFReflectionModelASF, BFReflectionAttrASF, SurfaceApproxASF, SurfaceEdgesASF, SurfaceEdgeTypeASF, SurfaceEdgeWidthASF, SurfaceEdgeColorASF*}
ASF_SPECIFIER	: [enables, asfs : BITMASK]
ASF_VALUE	: {*Bundled, Individual*}
ATEXT_STYLE	: ENUM_TYPE_INDEX (used with *ATextStyle* enumerated type)
BITMASK	: CARD32
BITMASK_SHORT	: CARD16
BOOLEAN	: {*False, True*}
BUFFER_MODE	: {*Single, Double*}
CARD8	: unsigned 8-bit integer
CARD16	: unsigned 16-bit integer
CARD32	: unsigned 32-bit integer
CHARACTER	: {CARD8, CARD16, CARD32}
COLOR	: {TABLE_INDEX, DIRECT_COLOR†}
CLIP_INDICATOR	: {*Clip, NoClip*}
COLOR_APPROX_MODEL	: ENUM_TYPE_INDEX (used with *ColorApproxModel* enumerated type)
COLOR_APPROX_TYPE	: ENUM_TYPE_INDEX (used with *ColorApproxType* enumerated type)
COLOR_MODEL	: ENUM_TYPE_INDEX (used with *RenderingColorModel* enumerated type)
COLOR_SPECIFIER	: [color_type : COLOR_TYPE, color_value : COLOR]
COLOR_TYPE	: ENUM_TYPE_INDEX (used with *ColorType* enumerated type)
COMPOSITION	: {*PreConcatenate, PostConcatenate, Replace*}
CONSTANT_NAME	: CARD16
CONTOUR	: {*Disjoint, Nested, Intersecting, Unknown*}
COORD	: {COORD_2D, COORD_3D, COORD_4D}
COORD_2D	: [x, y : FLOAT]
COORD_3D	: [x, y, z : FLOAT]
COORD_4D	: [wx, wy, wz, w : FLOAT]
COORD_TYPE	: {*Rational, NonRational*}
CULL_MODE	: {*None, BackFaces, FrontFaces*}
CURVE_APPROX	: [approx_method : CURVE_APPROX_METHOD, tolerance : FLOAT]
CURVE_APPROX_METHOD	: ENUM_TYPE_INDEX (used with *CurveApproxMethod* enumerated type)
DC_HIT_BOX_DATA	: [pick_position: DEVICE_COORD_2D, pick_distance: FLOAT]
DEVICE_COORD	: [x, y : INT16, z : FLOAT]
DEVICE_COORD_2D	: [x, y : INT16]
DEVICE_RECT	: [xmin, ymin, xmax, ymax : INT16]
DIRECT_COLOR	: direct color value†
DISPLAY_STATE	: {*NotEmpty, Empty*}

† See the "Extension Information" section for a list of the registered color types.

DISPLAY_UPDATE	: ENUM_TYPE_INDEX (used with *DisplayUpdateMode* enumerated type)
DRAWABLE_ID	: {WINDOW_ID, PIXMAP_ID}
DYNAMIC_TYPE	: {*IMM, IRG, CBS*}
ECHO_MODE	: {*NoEcho, Echo, Unecho*}
EDGE	: OPT_SWITCH
EDIT_MODE	: {*StructureInsert, StructureReplace*}
ELEMENT_INFO	: [type : ELEMENT_TYPE, length : CARD16]
ELEMENT_POS	: [whence : {*Beginning, Current, End*}, offset : INT32]
ELEMENT_RANGE	: [position1, position2 : ELEMENT_POS]
ELEMENT_REF	: [structure_id : STRUCTURE_ID, offset : CARD32]
ELEMENT_TYPE	: CARD16
ENUM_TYPE	:{*MarkerType, ATextStyle, InteriorStyle, HatchStyle,* *LineType, SurfaceEdgeType, PickDeviceType,* *PolylineInterpMethod, CurveApproxMethod, ReflectionModel,* *SurfaceInterpMethod, SurfaceApproxMethod,* *ModelClipOperator, LightType, ColorType,* *FloatFormat, HLHSRMode, PromptEchoType,* *DisplayUpdateMode, ColorApproxType, ColorApproxModel,* *GDP, GDP3, GSE, TrimCurveApproxMethod,* *RenderingColorModel, ParametricSurfaceCharacteristics,* *Escape, PickOneMethod, PickAllMethod*}
ENUM_TYPE_INDEX	: INT16
EXTENT_INFO	: [lower_left : COORD_2D, upper_right : COORD_2D, concatpoint : COORD_2D]
FACET	: [facet_data : OPT_DATA, vertices : LISTofVERTEX]
FLOAT	: floating point value†
FLOAT_FORMAT	: ENUM_TYPE_INDEX (used with *FloatFormat* enumerated type)
FONT_ID	: {PEX_FONT_ID, X11_FONT_ID}
HALFSPACE	: [point : COORD_3D, vector : VECTOR_3D]
HALFSPACE_2D	: [point : COORD_2D, vector : VECTOR_2D]
HATCH_STYLE	: ENUM_TYPE_INDEX (used with *HatchStyle* enumerated type)
HLHSR_MODE	: ENUM_TYPE_INDEX (used with *HLHSRMode* enumerated type)
INT8	: signed 8-bit integer
INT16	: signed 16-bit integer
INT32	: signed 32-bit integer
INTERIOR_STYLE	: ENUM_TYPE_INDEX (used with *InteriorStyle* enumerated type)
ISTRING	: LISTofMONO_ENCODING
LIGHT_TYPE	: ENUM_TYPE_INDEX (used with *LightType* enumerated type)
LINE_TYPE	: ENUM_TYPE_INDEX (used with *LineType* enumerated type)
LOOKUP_TABLE_ID	: RESOURCE_ID
MARKER_TYPE	: ENUM_TYPE_INDEX (used with *MarkerType* enumerated type)
MATCH_DRAW_TYPE	:{*DontCare, Window, Pixmap, Buffer*}
MATRIX	: FLOAT[4][4]‡

† See the "Extension Information" section for a list of the registered floating point formats.

MATRIX_3X3	: FLOAT[3][3]‡
MONO_ENCODING	: [char_set : CARD16,
	char_set_width : {*csByte*, *csShort*, *csLong*},
	encoding_state : CARD8,
	string : LISTofCHARACTER]
NAME	: CARD32
NAME_SET_ID	: RESOURCE_ID
NAME_SET_PAIR	: [incl: NAME_SET_ID,
	excl: NAME_SET_ID]
NPC_SUBVOLUME	: [min : COORD_3D,
	max : COORD_3D]
OPERATOR	: ENUM_TYPE_INDEX (used with *ModelClipOperator* enumerated type)
OPT_COLOR	: optional COLOR†
OPT_DATA	: [color : OPT_COLOR,
	normal : OPT_NORMAL,
	edge : OPT_SWITCH]
OPT_NORMAL	: optional VECTOR_3D†
OPT_SWITCH	: optional SWITCH†
OUTPUT_CMD	: [element_type : ELEMENT_TYPE,
	size : CARD16,
	data : ‡]
PC_BITMASK	: CARD32[3]
PEX_FONT_ID	: RESOURCE_ID
PHIGS_WKS_ID	: RESOURCE_ID
PIPELINE_CONTEXT_ID	: RESOURCE_ID
PICK_ALL_METHOD	: ENUM_TYPE_INDEX (used with *PickAllMethod* enumerated type)
PICK_ALL_STATE	: {MoreHits, NoMoreHits, MayBeMoreHits}
PICK_DATA	: {DC_HIT_BOX_DATA, NPC_SUBVOLUME}
PICK_DEVICE_TYPE	: ENUM_TYPE_INDEX (used with *PickDeviceType* enumerated type)
PICK_ELEMENT_REF	: [s_id : STRUCTURE_ID, *
	offset : CARD32,
	pickid : CARD32]
PICK_MEASURE_ID	: RESOURCE_ID
PICK_ONE_METHOD	: ENUM_TYPE_INDEX (used with *PickOneMethod* enumerated type)
PICK_RECORD	: [pick_type: PICK_DEVICE_TYPE,
	hit_box: PICK_DATA]
PICK_STATUS	: {NoPick, Ok, Aborted}
PIXMAP_ID	: RESOURCE_ID
POLYLINE_INTERP	: ENUM_TYPE_INDEX (used with *PolylineInterpMethod* enumerated type)
PROMPT_ECHO_TYPE	: ENUM_TYPE_INDEX (used with *PromptEchoType* enumerated type)
PSC_TYPE	: ENUM_TYPE_INDEX (used with *ParametricSurfaceCharacteristics*
	enumerated type)
PSURF_CHAR	: [psc_type : PSC_TYPE,
	psc_data : LISTofCARD8]
REFLECTION_ATTR	: [ambient_coef : FLOAT,
	diffuse_coef : FLOAT,

‡ Matrices are effectively passed as one-dimensional arrays of floating point values. For a 4×4 matrix, the matrix element used to represent the x translation value will be the fourth element in the array, the element containing the y translation value will be the eighth element, etc. 3×3 matrices are handled analogously.

† Indicates a parameter (or portion of a parameter) that may or may not be present in the request. However, its presence or absence can always be inferred from previous parameters in the request.

‡ See Section 3 - *Output Commands* for a description of each of the data records that can be passed/returned as an output command.

```
                         specular_coef : FLOAT,
                         specular_conc : FLOAT,
                         transmission_coef : FLOAT,
                         specular_color : COLOR_SPECIFIER]
REFLECTION_MODEL      : ENUM_TYPE_INDEX (used with ReflectionModel enumerated type)
RENDERER_ID           : RESOURCE_ID
RENDERER_STATE        : {Idle, Rendering, Picking}                              |
RENDERER_TARGET       : [depth: CARD8,                                          |
                         type: MATCH_DRAW_TYPE,                                 |
                         visual: VISUAL_ID ]                                    |
RESOURCE_ID           : 32-bit identifier
SEARCH_CONTEXT_ID     : RESOURCE_ID
SHAPE                 : {Convex, Nonconvex, Complex, Unknown}
STRING                : LISTofCARD8
STRUCTURE_ID          : RESOURCE_ID
STRUCTURE_INFO        : [id: RESOURCE_ID,
                         priority: FLOAT]
SURFACE_APPROX        : [approx_method : SURFACE_APPROX_METHOD,
                         u_tolerance, v_tolerance : FLOAT]
SURFACE_APPROX_METHOD : ENUM_TYPE_INDEX (used with SurfaceApproxMethod enumerated type)
SURFACE_EDGE_TYPE     : ENUM_TYPE_INDEX (used with SurfaceEdgeType enumerated type)
SURFACE_INTERP        : ENUM_TYPE_INDEX (used with SurfaceInterpMethod enumerated type)
SWITCH                : {Off, On}
TABLE_ENTRY           : [data : * ]
TABLE_INDEX           : CARD16
TABLE_TYPE            : {LineBundle, MarkerBundle, TextBundle, InteriorBundle,
                         EdgeBundle, Pattern, TextFont, Color, View, Light,
                         DepthCue, ColorApprox}
TEXT_ALIGNMENT        : [vertical : TEXT_VALIGNMENT,
                         horizontal : TEXT_HALIGNMENT]
TEXT_HALIGNMENT       : {HalignNormal, HalignLeft, HalignRight, HalignCenter}
TEXT_PATH             : {PathRight, PathLeft, PathUp, PathDown}
TEXT_PRECISION        : {String, Char, Stroke}
TEXT_VALIGNMENT       : {ValignNormal, ValignTop, ValignCap, ValignHalf,
                         ValignBase, ValignBottom}
TRIM_CURVE            : [visibility : SWITCH,
                         order : CARD16,
                         type : COORD_TYPE,
                         approx_method : TRIM_CURVE_APPROX_METHOD,
                         tolerance : FLOAT,
                         tmin, tmax : FLOAT,
                         knots : LISTofFLOAT,
                         points : LISTofCOORD]
TRIM_CURVE_APPROX_METHOD : ENUM_TYPE_INDEX (used with TrimCurveApproxMethod
                         enumerated type)
TYPE_OR_TABLE_INDEX   : {ENUM_TYPE_INDEX, TABLE_INDEX}
UPDATE_STATE          : {NotPending, Pending}
VECTOR_2D             : [x, y : FLOAT]
VECTOR_3D             : [x, y, z : FLOAT]
VERTEX                : [point : COORD_3D,
                         data : OPT_DATA]
```

* See the section "Lookup Tables" for a description of each of the data records that can be passed/returned as a table entry.

VIEWPORT	: [min : DEVICE_COORD, max : DEVICE_COORD, use_drawable : BOOLEAN]
VIEW_REP	: [index : TABLE_INDEX, clip_flags : BITMASK, clip_limits : NPC_SUBVOLUME, orientation : MATRIX, mapping : MATRIX]
VISUAL_ID	: RESOURCE_ID
VISUAL_STATE	: {*Correct*, *Deferred*, *Simulated*}
WINDOW_ID	: RESOURCE_ID
WKS_BITMASK	: CARD32[2]
X11_FONT_ID	: RESOURCE_ID

2.5.7. Errors

If an error occurs while processing a request that modifies a PEX resource then it is possible that some portion of the resource (or some subset of the resource attributes) will have been altered. |

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 following error codes can be returned by the various PEX requests:

ColorType
The specified color type is not supported.

FloatingPointFormat
The specified floating point format is not supported.

Label
The specified label does not exist in the structure.

LookupTable
A value for a lookup table argument is illegal or does not name a defined lookup table resource.

NameSet
A value for a name set argument is illegal or does not name a defined name set resource.

OutputCommand
A value for some parameter of an output command is illegal, out of range, or otherwise inappropriate.

Path
A value for a structure network path contains inappropriate or illegal values.

PEXFont
A value for a PEX font argument is illegal or does not name a defined PEX font resource.

PhigsWKS
A value for a PHIGS workstation argument is illegal or does not name a defined PHIGS workstation resource.

PickMeasure
A value for a pick measure argument is illegal or does not name a defined pick measure resource.

PipelineContext
A value for a pipeline context argument is illegal or does not name a defined pipeline context resource.

Renderer
A value for a renderer argument is illegal or does not name a defined renderer resource.

RendererState
A renderer was in the *Rendering* state when a **PEXBeginRendering** request was received.

SearchContext
A value for a search context argument is illegal or does not name a defined search context resource.

Structure
A value for a structure argument is illegal or does not name a defined structure resource.

2.6. Events

All PEX events will use the same mechanisms as X events. |

PEX defines a new event, **MaxHitsReached**. **PEXGetImpDepConstants** can be used to determine whether this event is supported. This event is returned during a client side pick all traversal to indicate that the maximum number of hits has been reached. The event includes the renderer identifier. Upon receiving the event, the client |

should terminate the pick traversal and decide whether an additional pass is necessary. |

2.7. Padding

Certain values that must line up on 2- or 4-byte boundaries may necessitate the insertion of pad bytes in some requests. The value of pad bytes is undefined.

2.8. Extension Information

These requests return static information about the PEX extension and what it supports. Information about specific capabilities and tradeoffs should be found in the documentation describing a particular PEX server implementation (e.g., what is the "best" HLHSR method or floating point format or direct color format to use, whether quick update really does anything, what range of line and surface edge widths are supported, etc.)

2.8.1. Get Extension Information

Name:

PEXGetExtensionInfo

Request:

client_protocol_major_version : CARD16
client_protocol_minor_version : CARD16

Reply:

protocol_major_version : CARD16
protocol_minor_version : CARD16
vendor : STRING
release_number : CARD32
subset_info : CARD32

Errors:

None

Description:

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 one. The *vendor* parameter is a string of ISO-LATIN1 characters that describes the vendor that supplied the PEX extension. The release number is a 32-bit value whose semantics are controlled by the vendor. The top 16 bits of *subset_info* are reserved for use by vendors and the bottom 16 bits contain information about whether the PEX server is a full PEX implementation or whether it supports some combination of the standard subsets.

If the 16 low-order bits of *subset_info* are zero, the extension is a complete PEX implementation. If the lowest-order bit of *subset_info* is set, then the PEX extension supports "immediate rendering". If the next-to-lowest-order bit of *subset_info* is set, then the PEX extension supports "PHIGS workstation". If the third-lowest bit of *subset_info* is set, then the PEX extension supports "structure rendering". If a server is sent a request that is not in the PEX subset supported by that server, it will return a *Request* error. See Appendix A for the definition of "immediate rendering", "PHIGS workstation", and "structure rendering" subsets.

The string "X3D-PEX" should be returned by the X request **ListExtensions** to indicate the presence of the PEX extension. The same string should be used by clients in the X request **QueryExtension**.

2.8.2. Get Enumerated Type Information

Name:

PEXGetEnumeratedTypeInfo

Request:

drawable_id : DRAWABLE_ID
enum_types : LISTofENUM_TYPE
item_mask : BITMASK

Reply:

types : LISTofLISTofVALUE

Errors:

Drawable, Match, Value |

Description:

This request returns information about the enumerated types specified by *enum_types*. It returns information about the enumerated types that are supported for drawables that have the same root window and depth as the drawable indicated by *drawable_id*. The *item_mask* indicates the data that is to be returned to describe each enumerated type. The components of an enumerated type descriptor (and the corresponding bits of *item_mask*) are:

index	: ENUM_TYPE_INDEX
mnemonic	: STRING

If only the *index* bit is set in *item_mask*, a list of index values (type ENUM_TYPE_INDEX) will be returned for the defined values for each enumerated type specified in the *enum_types* list. If only the *mnemonic* bit is set in *item_mask*, only descriptor strings that use the ISO-Latin1 encoding will be returned for the defined values (type STRING). If both the *index* and *mnemonic* bits are set, an index/mnemonic pair will be returned for each of the defined values for each of the requested enumerated types. If neither bit is set, a list of counts will be returned, where each count represents the number of supported types for each entry in *enum_types*.

The various enumerated types and registered values are listed below. Each registered value is followed by the mnemonic string that is returned and a brief description. Strings are returned using the ISO-Latin1 character set. The strings are returned exactly as shown below. Any enumerated type values less than zero are implementation-dependent (consult the implementation documentation for their descriptions), and any numbers greater than the listed values are reserved for future registration.

MarkerType

The marker type specifies the shape of the marker primitive that is to be drawn when rendering marker primitives. The registered values are:

1	Dot	"." which is always displayed as the smallest displayable dot (the *marker_scale* attribute is ignored) with the dot at the marker position.
2	Cross	"+" (cross or plus sign) with intersection at the marker position.
3	Asterisk	"*" with intersection at the marker position.
4	Circle	"o" with center at marker position.
5	X	"x" with intersection at the marker position.

ATextStyle

The annotation text style specifies the style that is to be used when rendering annotation text primitives. The registered values are:

1	NotConnected	The annotation text primitive will be drawn with no line connecting it to the annotation text reference point.
2	Connected	The annotation text primitive will be connected to the annotation text reference point with a line, which will be drawn using the current set of line attributes.

LineType *

The line type specifies the style that is to be used when rendering polyline and curve primitives. The registered values are:

1	Solid	Draw the polyline or curve with a solid, unbroken line.
2	Dashed	Draw the polyline or curve with a line that is dashed.
3	Dotted	Draw the polyline or curve with a line that is dotted.
4	DashDot	Draw the polyline or curve with a line that contains alternating dots and dashes.

It is implementation-dependent whether the sequence for the *Dashed*, *Dotted*, and *DashDot* 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.

PolylineInterpMethod *

The polyline interpolation method specifies the style that is to be used when rendering polyline primitives that have colors specified per-vertex. Depth-cueing is applied as a post-process to polylines regardless of the polyline interpolation method. The registered values are:

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 ith 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 nth will be ignored), or whether they will be used to compute an average color which will be used for the entire segment.
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.

CurveApproxMethod

The curve approximation method specifies the method that is to be used when rendering non-uniform rational B-spline (NURB) curve primitives. The registered values are:

1	(imp. dep.)	This value for *CurveApproxMethod* is supported on every implementation, but may differ from one to the next. It may have the same mnemonic and definition as one of the other types, or it may be a method that is not in the list of registered types.
2	ConstantBetweenKnots	This technique tessellates the curve with equal parametric

increments between successive pairs of knots. The tolerance value controls tesselation of the curve. If the tolerance value is not an integer value, it is truncated and only the integer portion will be 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	This technique tessellates the curve until the length of each line segment (chord) in world coordinates is less than the tolerance.
4	NPC_ChordalSize	This technique tessellates the curve until the length of each line segment (chord) in normalized project coordinates is less than the tolerance.
5	DC_ChordalSize	This technique tessellates the curve until the length of each line segment (chord) in device coordinates is less than the tolerance.
6	WCS_ChordalDev	This technique tessellates the curve until the maximum deviation (in world coordinates) between the line and the curve is less than the tolerance.
7	NPC_ChordalDev	This technique tessellates the curve until the maximum deviation (in normalized projection coordinates) between the line and the curve is less than the tolerance.
8	DC_ChordalDev	This technique tessellates the curve until the maximum deviation (in device coordinates) between the line and the curve is less than the tolerance.
9	WCS_Relative	This technique maintains a relative level of quality based on the tolerance value independent of scaling in world coordinates.
10	NPC_Relative	This technique maintains a relative level of quality based on the tolerance value independent of scaling in normalized projection coordinates.
11	DC_Relative	This technique maintains a relative level of quality based on the tolerance value independent of scaling in device coordinates.

InteriorStyle |

The interior style specifies the style that is to be used when rendering surface primitives. The registered values are: |

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. |

2 Solid — The interiors of surface primitives are filled using the surface color.

3 Pattern — The interiors of surface primitives are filled using the pattern table entry specified by the interior style index.

4 Hatch — The interiors of surface primitives are filled using the surface color and the hatch style whose index is specified by the interior style index.

5 Empty — The interior of the surface primitive is not drawn at all.

HatchStyle

The hatch style specifies the method that is to be used to render surface primitives when the interior style is set to *Hatch*. There are currently no registered hatch styles.

SurfaceEdgeType

The surface edge type specifies the style that is to be used when rendering surface edges. The registered values are:

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.

It is implementation-dependent whether the sequence for the *Dashed*, *Dotted*, and *DashDot* 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.

ReflectionModel

The reflection model specifies the method that is used to perform the light source shading computation when rendering surface primitives. The input to the light source shading computation is known as the *intrinsic color* and the output is known as the *shaded color*. If a normal exists at the point at which the reflection model is to be evaluated, it will be used. Otherwise, if a normal exists for the facet containing the point, it will be used to evaluate the reflection model. If no normal exists, the reflection model is evaluated, if possible, without a normal. The registered values are:

1 NoShading — No light source shading computation is performed. The surface color is not affected by light source illumination (effectively, shaded color ≡ intrinsic color).

2 Ambient — Only the ambient terms of the lighting equation are used. The shaded color will be the intrinsic color as seen under ambient light.

3 Diffuse — Only the ambient and diffuse terms of the lighting equation are used. The shaded color will be the intrinsic color as seen under ambient light, plus a diffuse reflection component from each light source.

4 Specular — The ambient, diffuse, and specular terms of the lighting

equation are all used during the light source shading computation. The shaded color will be the same as for *Diffuse*, plus a specular reflection component from each light source.

SurfaceInterpMethod

The surface interpolation method specifies the method that is used to compute color values in surface interiors when rendering surface primitives. Depth-cueing is applied as a post-process to surface primitives regardless of the surface interpolation method. The registered values are:

1	None	The color resulting from a single light source computation is used for the entire surface. No interpolation will be performed across surface interiors or edges.
2	Color	The colors are computed at the vertices of the surface according to the current *reflection_model*. These color values are then linearly interpolated across the interior of the surface or the edges.
3	DotProduct	The lighting equation dot products are computed at the vertices. These dot products are linearly interpolated and the light source shading computation is applied using these values to compute the color value at each pixel in the interior of a surface or along a surface edge.
4	Normal	An attempt is made to interpolate the normal across the facet and perform the light source shading computation as accurately as possible at each pixel in the interior of a surface or along a surface edge.

SurfaceApproxMethod

The surface approximation method specifies how to display non-uniform rational B-spline surface primitives. The registered values are:

1	(imp. dep.)	This value for *SurfaceApproxMethod* is supported on every implementation, but may differ from one to the next. It may have the same mnemonic and definition as one of the other types, or it may be a method that is not in the list of registered types.
2	ConstantBetweenKnots	This technique tessellates the surface with equal parametric increments between successive pairs of knots. The two tolerance values control tesselation in each of the two parameter dimensions. If the tolerance values are not integer values, they are truncated and only the integer portions of each will be used. If *u_tolerance* is less than or equal to zero, the surface will be evaluated only at the *u* parameter limits in the *u* direction, and at the *u* knots that are within the specified parameter range. If *u_tolerance* is greater than zero, the surface will be evaluated at the *u* parameter limits in the *u* direction, at the *u* knots that are within the specified parameter range, and at the number of positions specified by *u_tolerance* between each pair of knots. The

		value of *v_tolerance* is used similarly to control the evaluation in the *v* direction.
3	WCS_ChordalSize	This technique tessellates the surface until the length of each line segment (chord) in world coordinates in the *u* parameter direction is less than the specified *u* tolerance value, and the length of every line segment in world coordinates in the *v* parameter direction is less than the specified *v* tolerance value.
4	NPC_ChordalSize	This technique tessellates the surface until the length of each line segment (chord) in normalized projection coordinates in the *u* parameter direction is less than the specified *u* tolerance value, and the length of every line segment in normalized projection coordinates in the *v* parameter direction is less than the specified *v* tolerance value.
5	DC_ChordalSize	This technique tessellates the surface until the length of each line segment (chord) in device coordinates in the *u* parameter direction is less than the specified *u* tolerance value, and the length of every line segment in device coordinates in the *v* parameter direction is less than the specified *v* tolerance value.
6	WCS_PlanarDev	This technique tessellates the surface into facets. The technique subdivides the surface until the absolute value of the maximum deviation, in world coordinates, between any facet and the surface is less than *u_tolerance*.
7	NPC_PlanarDev	This technique tessellates the surface into facets. The technique subdivides the surface until the absolute value of the maximum deviation, in normalized projection coordinates, between any facet and the surface is less than *u_tolerance*.
8	DC_PlanarDev	This technique tessellates the surface into facets. The technique subdivides the surface until the absolute value of the maximum deviation, in device coordinates, between any facet and the surface is less than *u_tolerance*.
9	WCS_Relative	This technique maintains a relative level of quality based on the specified *u_tolerance* value independent of scaling in world coordinates.
10	NPC_Relative	This technique maintains a relative level of quality based on the specified *u_tolerance* value independent of scaling in normalized projection coordinates.
11	DC_Relative	This technique maintains a relative level of quality based on the specified *u_tolerance* value independent of scaling in device coordinates.

TrimCurveApproxMethod

The trim curve approximation method specifies the method that is to be used for trim curves when rendering non-uniform rational B-spline (NURB) surface primitives with trim curves. The registered values are:

1	(imp. dep.)	This value for *TrimCurveApproxMethod* is supported on every implementation, but may differ from one to the next. It may have the same mnemonic and definition as one of the other types, or it may be a method that is not in the list of registered types.
2	ConstantBetweenKnots	This technique tessellates the trim curve with equal parametric increments between successive pairs of knots. The tolerance value controls tesselation of the trim curve. If the tolerance value is not an integer value, it is truncated and only the integer portion will be 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.

ModelClipOperator

The model clip operator defines the operation that is to be used to combine the specified halfspaces with the current composite modeling clipping volume. The registered values are:

1	Replace	The specified halfspaces are used to create a new composite modeling clipping volume that replaces the current composite modeling clipping volume.
2	Intersection	The specified halfspaces are intersected with the current composite modeling clipping volume to compute a new composite modeling clipping volume.

LightType

The light type defines the characteristics of the light sources that can be used in light source shading computations. The registered values are:

1	Ambient	A light source that affects all surface primitives uniformly. Ambient light sources have only a color attribute.
2	WCS_Vector	A light source that is specified in world coordinates with a color and a direction vector.
3	WCS_Point	A light source that is specified in world coordinates with a color, a position, and two attenuation coefficients.
4	WCS_Spot	A light source that is specified in world coordinates with a color, a position, a direction vector, a concentration exponent, two attenuation coefficients and a spread angle.

ColorType

The color type defines the format of color values. The registered values are:

0	Indexed	A color that is passed as an unsigned 16-bit integer (i.e.,

it is of type TABLE_INDEX). The integer value is used as an index into a color lookup table. Dereferencing of an indexed color value occurs at the time of rendering, at the time when the actual color value is needed for rendering an output primitive.

1	RGBFloat	A color that is passed as three floating point values, in the order red [0-1], green [0-1], blue [0-1]. A color in this format has a type defined by: COLOR_RGB_FLOAT: [r, g, b : FLOAT]
2	CIEFloat	A color that is passed as three floating point values, in the order u [0-1], v [0-1] (CIELUV diagram coefficients), and luminance [0-1]. A color in this format has a type defined by: COLOR_CIE_FLOAT: [u, v, luminance : FLOAT]
3	HSVFloat	A color that is passed as three floating point values, in the order hue [0-1] (angle in fractions of a circle, with red being zero), saturation [0-1], and value [0-1]. A color in this format has a type defined by: COLOR_HSV_FLOAT: [hue, saturation, value : FLOAT]
4	HLSFloat	A color that is passed as three floating point values, in the order hue [0-1] (angle in fractions of a circle, with red being zero), lightness [0-1], and saturation [0-1]. A color in this format has a type defined by: COLOR_HLS_FLOAT: [hue, lightness, saturation : FLOAT]
5	RGBInt8	A color that is passed as a unit of four bytes, in the order red, green, blue. A color in this format has a type defined by: COLOR_RGB_INT8: [r, g, b, pad : CARD8]
6	RGBInt16	A color that is passed as a unit of eight bytes, in the order red, green, blue. A color in this format has a type defined by: COLOR_RGB_INT16: [r, g, b, pad : CARD16]

FloatFormat

The floating point format defines the format of floating point values. The registered values are:

1	IEEE_754_32	An IEEE 754 standard 32-bit floating point value.
2	DEC_F_Floating	A DEC F-floating value.
3	IEEE_754_64	An IEEE 754 standard 64-bit floating point value.
4	DEC_D_Floating	A DEC D-floating value.

HLHSRMode

The HLHSR mode defines the method used to do hidden line/hidden surface removal. The registered values are:

1	Off	All output primitives are drawn in the order they are processed. No attempt will be made to remove hidden surfaces.

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 device-dependent. This technique permits visibility to be computed without an intermediate storage area for transformed data, can be used to incrementally add primitives to an image, and is an HLHSR method which is of linear order.
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. This technique is fairly fast for small numbers of primitives, but requires an intermediate storage area. This technique does not guarantee totally correct results, since it fails in cases involving cyclically overlapping or interpenetrating objects, and in other, even simpler, cases.
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. This technique can be fairly fast for small numbers of polygons, but uses an intermediate storage area to buffer output primitives and must perform a sorting step.
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 a hidden line computation is performed.
6	ZBufferId	This is the same as *ZBuffer*, except that the *HLHSR_identifier* is used to enable and disable z-buffering during traversal. An *HLHSR_id* of zero disables z-buffering and an *HLHSR_identifier* of one enables z-buffering.

PickDeviceType

The pick device type specifies the type of pick device that is to be used for workstation picking (via a pick device and a pick measure) or renderer picking. If a pick measure is created using one of the registered pick device types then the *pick_data_rec* component of the pick device descriptor is ignored and the default prompt and echo type is *EchoPrimitive*. The *pick_data_rec* component of the pick device descriptor is used in conjunction with non-registered pick device types. The registered values are:

1	DC_HitBox	The pick aperature is specified by a pick position and a pick distance, both in device coordinates. The shape of the hit box (square, circle, etc.) is implementation-dependent. The pick distance defines the half-width or radius of the hit box.
2	NPC_HitVolume	The pick aperature is specified by two points that describe a parallelpiped in NPC space.

PickOneMethod

The pick one method specifies the pick criteria that is to be used for renderer pick one traversals. The registered values are:

1	Last	The last picked primitive is returned
2	ClosestZ	The primitive which has the z value closest to the front clipping plane in the pick aperature is returned. If several primitives are equally close to the front clipping plane then any of these primitives may be returned.
3	VisibleAny	Any picked primitive which is visible (after taking the current HLHSR mode into account) may be returned.
4	VisibleClosest	The primitive which is both visible (after taking the current HLHSR mode into account) and closest to the pick position is returned. The algorithm for determining which primitive is closest to the pick position is implementation dependent.

PickAllMethod

The pick all method specifies the pick criteria that is to be used for renderer pick all traversals. The registered values are:

1	All	All primitives which are contained in or intersect the pick aperature are returned.
2	Visible	Only visible primitives which are contained in or intersect the pick aperature are returned.

PromptEchoType

The prompt echo type defines the method used to do prompting and echoing during picking operations. The registered values are:

1	EchoPrimitive	Use an implementation-dependent technique that at least highlights the picked primitive for a short period of time.
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).
3	EchoNetwork	Echo the entire posted structure network that contains the picked primitive.

DisplayUpdateMode

The display update mode defines the manner in which changes will affect the displayed image. The registered values are:

1	VisualizeEach	Visualize each change as it occurs. (PHIGS - ASAP)
2	VisualizeEasy	Visualize only the changes that are "easy to do" (PHIGS -

		WAIT/UWOR). Things that are "easy to do" are those that have a dynamic modification of *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.
3	VisualizeNone	Visualize none of the changes (PHIGS - WAIT/NIVE). The changes are applied, but the image is not regenerated until there is an explicit request to do so.
4	SimulateSome	Visualize the easy changes and simulate those changes that can be simulated. (PHIGS - WAIT/UQUM)
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)

It should be noted that 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.

ColorApproxType

The color approximation type describes the way that a renderer will transform rendering pipeline color values into displayable pixel values. The registered values are:

1	ColorSpace	The rendering pipeline color is converted into a color with three individual color components.
2	ColorRange	The rendering pipeline color is converted into a single color index.

This enumerated type allows applications to control whether the color value produced through illumination and depth-cueing computations is transformed into a single value (e.g., for display on an 8-bit pseudo color display) or into three values (e.g., for display on a 24-bit direct color display).

ColorApproxModel

The color approximation model describes the space in which any color filtering or sampling will be performed during the color approximation phase of rendering. The registered values are:

1	RGB	red, green, blue
2	CIE	CIELUV diagram u, v coordinates plus luminance
3	HSV	hue, saturation, value
4	HLS	hue, lightness, saturation
5	YIQ	(NTSC) luminance (Y), inphase (wideband orange-cyan), and quadrature (narrowband magenta-green)

RenderingColorModel *

The rendering color model defines the color model to be used for color interpolation within the rendering pipeline. Reflectance equations should have the appearance of being performed in the color space specified by the rendering color model.

0	(imp. dep.)	An implementation-dependent color space
1	RGB	red, green, blue color model
2	CIE	CIELUV diagram u, v coordinates plus luminance color model
3	HSV	hue, saturation, value color model
4	HLS	hue, lightness, saturation color model

ParametricSurfaceCharacteristics

1 None — No additional surface characteristics beyond the current surface attributes

2 (imp. dep.) — An implementation-dependent method that displays the shape of the surface. This method does not distinguish between front and 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. The data record is ignored for this type.

3 IsoparametricCurves — Isoparametric curves are drawn on the surface. 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. This method does not distinguish between front and 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.

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 modelling coordinate direction vector. The positions of the planes are specified by a sequence of intersection points along an infinite line defined by a modelling coordinate origin point, P_0, and a direction vector, $\vec{V}$.

$$P_i = P_0 + t_i \vec{V}$$

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 modelling coordinates, and $\vec{V}$ is the specified direction vector in modelling coordinates. The i-th plane is perpendicular to the direction vector, $\vec{V}$, and intersects the infinite line at point P_i. The data record consists of the origin point, P_0; the direction vector, $\vec{V}$; and the list of parameters, t_i.

This method does not distinguish between front and 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 the primitive's filled or hollow interiors, but lower priority than the primitive's edges.

5 WC_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 modelling coordinate direction vector. The positions of the planes are specified by a sequence of intersection points along an infinite line defined by a modelling coordinate origin point, P_0, and a direction vector, $\vec{V}$.

$$P_i = P_0 + t_i \vec{V}$$

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 world coordinates, and $\vec{V}$ is the specified direction vector in world coordinates. The i-th plane is perpendicular to the direction vector, $\vec{V}$, and intersects the infinite line at point P_i. The data record consists of the origin point, P_0; the direction vector, $\vec{V}$; and the list of parameters, t_i.

This method does not distinguish between front and 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 the primitive's filled or hollow interiors, but lower priority than the primitive's edges.

GDP

The GDP type specifies the (2D) Generalized Drawing Primitives (GDPs) that are supported by the PEX extension implementation. There are currently no registered GDPs.

GDP3

The GDP3 type specifies the (3D) Generalized Drawing Primitives (GDP3s) that are supported by the PEX extension implementation. There are currently no registered GDP3s.

GSE

The GSE type specifies the Generalized Structure Elements (GSEs) that are supported by the PEX extension implementation. There are currently no registered GSEs.

Escape

The Escape type specifies the Escape Requests that are supported by the PEX extension implementation. The registered escapes are described in appendix E, they are:

1	SetEchoColor	modify the renderer's echo color

2.8.3. Get Implementation-Dependent Constants

Name:

PEXGetImpDepConstants

Request:

fp_format : FLOAT_FORMAT
drawable_example : DRAWABLE_ID
names : LISTofCONSTANT_NAME

Reply:

constants : LISTofVALUE

Errors:

Drawable, Match, Value, FloatingPointFormat |

Description:

This request allows a client to query one or more of the implementation-dependent constants in a PEX server extension. A single CARD32 or FLOAT is returned for each value requested. These values are returned in order, with one return value in *constants* for each requested value in *names*. Floating-point values will be returned in the format specified by *fp_format*. The implementation-dependent constants that are returned are based on the values that would be used for a drawable with the same root and depth as *drawable_example*.

PEX defines a number of standard constant names that all PEX extensions must be able to return. These standard constant names are 16-bit integers with the high order bit equal to zero. Additional proprietary implementation-dependent constants can be defined and returned by PEX server extensions using 16-bit integers with the high order bit equal to one. The standard constant names consist of:

NominalLineWidth	CARD32	Width (in pixels) of "standard" line or curve.
NumSupportedLineWidths	CARD32	Number of supported line or curve widths (a value of 0 indicates that all line widths, including fractional widths, between min and max line width are supported).
MinLineWidth	CARD32	Width (in pixels) of thinnest line or curve that can be drawn.
MaxLineWidth	CARD32	Width (in pixels) of thickest line or curve that can be drawn.
NominalEdgeWidth	CARD32	Width (in pixels) of "standard" edge.
NumSupportedEdgeWidths	CARD32	Number of supported edge widths (a value of 0 indicates that all edge widths, including fractional widths, between min and max edge width are supported).
MinEdgeWidth	CARD32	Width (in pixels) of thinnest edge that can be drawn.
MaxEdgeWidth	CARD32	Width (in pixels) of thickest edge that can be drawn.
NominalMarkerSize	CARD32	Largest dimension (either height or width, in pixels) of "standard" marker.
NumSupportedMarkerSizes	CARD32	Number of supported marker sizes (a value of 0 indicates that all marker sizes, including fractional values, between min and max marker size are supported).

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).
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).
ChromaticityRedU	FLOAT	Returns the CIELUV *u* chromaticity coefficient for the red channel of the (properly adjusted) display device.
ChromaticityRedV	FLOAT	Returns the CIELUV *v* chromaticity coefficient for the red channel of the (properly adjusted) display device.
LuminanceRed	FLOAT	Returns the CIELUV luminance value for the red channel of the (properly adjusted) display device.
ChromaticityGreenU	FLOAT	Returns the CIELUV *u* chromaticity coefficient for the green channel of the (properly adjusted) display device.
ChromaticityGreenV	FLOAT	Returns the CIELUV *v* chromaticity coefficient for the green channel of the (properly adjusted) display device.
LuminanceGreen	FLOAT	Returns the CIELUV luminance value for the green channel of the (properly adjusted) display device.
ChromaticityBlueU	FLOAT	Returns the CIELUV *u* chromaticity coefficient for the blue channel of the (properly adjusted) display device.
ChromaticityBlueV	FLOAT	Returns the CIELUV *v* chromaticity coefficient for the blue channel of the (properly adjusted) display device.
LuminanceBlue	FLOAT	Returns the CIELUV luminance value for the blue channel of the (properly adjusted) display device.
ChromaticityWhiteU	FLOAT	Returns the CIELUV *u* chromaticity coefficient for the reference white of the (properly adjusted) display device.
ChromaticityWhiteV	FLOAT	Returns the CIELUV *v* chromaticity coefficient for the reference white of the (properly adjusted) display device.
LuminanceWhite	FLOAT	Returns the CIELUV luminance value for the reference white of the (properly adjusted) display device.
MaxNamesetNames	CARD32	Maximum number of names allowed in a name set. The names are between zero and (*MaxNamesetNames* - 1) |
MaxModelClipPlanes	CARD32	Maximum number of modeling clipping planes that may be defined.
TransparencySupported	CARD32	Returns *True* or *False*, depending on whether the transmission coefficient is utilized in the light source shading computations.

DitheringSupported	CARD32	Returns *True* or *False*, depending on whether the dithering hint actually causes dithering to occur.
MaxNonAmbientLights	CARD32	Maximum number of non-ambient light sources that may be enabled at one time.
MaxNURBOrder	CARD32	Maximum non-uniform rational B-spline order supported.
MaxTrimCurveOrder	CARD32	Maximum order for trim curves.
BestColorApproxValues	CARD32	Returns the constant *ColorApproxPowersOf2* to indicate whether it is a significant performance win if the color approximation multiplier values are powers of two so that pixels can be composed using shifts and adds, or *ColorApproxAnyValues* if it makes little or no difference.
DoubleBufferingSupported	CARD32	Returns *True* or *False* depending on whether or not the server supports double-buffering.
MaxHitsEventSupported	CARD32	Returns *True* or *False*, depending on whether the *MaxHitsReached* event is supported.

2.8.4. Match Renderer Targets

Name:

PEXMatchRendererTargets

Request:

drawable_id : DRAWABLE_ID
depth : CARD8
type : MATCH_DRAW_TYPE
visual : VISUAL_ID
max_targets : CARD32

Reply:

targets : LISTofRENDERER_TARGETS

Errors:

Drawable, Value

Description:

This returns information about the visual, depth and drawables the server can support. *drawable_id* tells the server which screen the client is interested in and, *depth*, *visual* and *type* specify which data is of interest. Wildcarding can be accomplished by specifying zero for the *depth*, zero for the *visual* and *DontCare* for the *type*. This matches all depths, visuals and/or drawable types. *max_targets* is the maximum number of targets the client wants returned.

2.8.5. Escape

Name:

PEXEscape

Request:

escape_id : CARD32
escape_data : LISTofCARD8

Errors:

Value, *escape-specific*

Description:

This request can be used to send registered PEX escapes or vendor-specific escapes that do not generate a reply. Registered escapes are defined in Appendix E. For registered escapes, *escape_id* is a positive number. For vendor-specific escapes the *escape_id* is encoded as follows: the high bit is set, the next 15 bits contain the *vendor_id* and the low-order 16 bits contain the vendor-specific *request_id*. *vendor_id* is a unique number, assigned on request by the MIT registry, that identifies the vendor. The supported escapes can be determined using **PEXGetEnumeratedTypeInfo**.

For vendor-specific escapes, it is up to the vendor to specify how *escape_data* is encoded. For example, one vendor may choose to always pass a floating point format as the first value in *escape_data* while another may choose to pass a sub-request number. As with all X requests, this request must be a multiple of 4 bytes.

If the *escape_id* is not recognized by the server, or if the *escape_id* refers to an escape that generates a reply, a Value error will be generated. All other errors returned by this request are escape-specific. This request will not generate any new events or errors. The X extension mechanism must be used for vendor-specific requests that generate new events or new errors.

2.8.6. Escape With Reply

Name:

PEXEscapeWithReply

Request:

escape_id : CARD32
escape_data : LISTofCARD8

Reply:

escape_id : CARD32
escape_reply_data : LISTofCARD8

Errors:

Value, *escape-specific*

Description:

This request can be used to send registered PEX escapes or vendor-specific escapes that generate a reply. The supported escapes can be determined using **PEXGetEnumeratedTypeInfo**. At this point in time there are no registered escapes that have replies.

For vendor-specific escapes, it is up to the vendor to specify how *escape_data* and *escape_reply_data* are encoded. The vendor should define encodings which follow the conventions in the encoding document.

The encoding of *escape_id* is described in **PEXEscape**. If the *escape_id* is not recognized by the server, or if the *escape_id* refers to an escape that does not generate a reply, a Value error will be generated. All other errors returned by this request are escape-specific. This request will not generate any new events or errors. The X extension mechanism must be used for vendor-specific requests that generate new events or new errors.

3. Output Commands

This section defines output commands. Output commands are commands that are capable of being processed by a renderer or stored as structure elements. The format of each of the commands is listed below. Output commands may be passed to the server to be processed immediately by a renderer with the **PEXRenderOutputCommands** request. Output commands may be passed to the server to be stored in a structure with the **PEXStoreElements** request. Output commands may be retrieved by a client from a structure resource with the **PEXFetchElements** request.

Output commands are always executed in exactly the same fashion, no matter whether they are processed immediately by a renderer or processed as part of a structure traversal. When sent to the server via a **PEXRenderOutputCommands** request, output commands are processed until one is found to be in error, or until the entire list has been processed. If an output command is discovered to contain an error, it is discarded, as are all others following it in the list of output commands and an *OutputCommand* error is returned to the client. Similarly, if a **PEXStoreElements** command is used to transmit a list of output commands to the server, the first erroneous output command and all output commands following it in the list will be discarded, and an *OutputCommand* error will be reported to the client. Thus it is not possible for a structure resource to contain any elements with illegal or inappropriate values.

OutputCommand errors are only generated by the **PEXRenderOutputCommands** or **PEXStoreElements** requests. The error checking performed for these two types of requests is identical. When output commands are processed by a renderer, attribute specifications that are not supported by the server, out-of-range table indices, or undefined table indices are mapped to their default values; no errors will be reported.

3.1. Data Formats

Each of the requests listed above takes a format parameter of type FLOAT_FORMAT. For those requests sending data from the client to the server, this format word is used to indicate to the server the format of any floating point values that are contained in the request. For those requests requiring data to be sent back to the client, the format is used to indicate to the server how it should format the floating point data in the reply sent back to the client.

Color values are typically passed as a color type and a value. The color type specifies whether the color is an index value or a direct color value of some type. In the case of the "with data" output primitives (which may contain many color values), the color type is specified just once and all of the color values in the output command must be of the indicated color type. PEX servers are required to be able to deal with indexed colors and at least one type of direct color. Indexed colors are specified using an index which is used to obtain the color from a color lookup table. Direct colors are specified directly as RGB, HSV, HLS, or CIELUV color values of some form. The list of registered direct color formats can be found in the "Extension Information" section. PEX servers are free to store direct color values in whatever implementation-dependent format they choose, but they must be capable of converting those values back into the originally-specified color type when queried by the client.

3.2. Errors

Errors that can be reported when passing a list of output commands to a PEX server are described in the following sections. Specific errors that the PEX server checks for when storing or processing a particular type of output command are explained in the "Output Command Descriptions" section.

3.2.1. FloatingPointFormat Errors

The floating point format is specified once in each **PEXStoreElements** or **PEXRenderOutputCommands** request. If the 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 are processed, even if they do not contain floating point values. All output commands in the list are ignored.

3.2.2. ColorType Errors

The color type is generally specified with each color or, in some cases, once per output command. If an output command specifies a color type that is not supported by the PEX server, an *OutputCommand* error is reported. All previous output commands in the list are processed, and the output command containing the unsupported color type and any subsequent output commands are ignored.

3.2.3. Length Errors

If an output command exceeds the length of the output command list, processing of the list stops and an *OutputCommand* error is reported. All previous output commands in the list are processed, and the output command that exceeds the request length and any subsequent output commands are ignored.

3.2.4. OutputCommand Errors

If an illegal value is specified in an output command, all processing of the list stops and an *OutputCommand* error is reported. All previous output commands in the list are processed, and the output command that contains the illegal value and any subsequent output commands are ignored.

Not all unsupported values are illegal. Some enumerated types allow for implementation dependent values (for example, negative line types). In general, output commands that contain these types can have arbitrary values specified. When the output command is rendered, values that are not supported by the PEX server are rendered with a default value and do not report errors.

A second category of enumerated types have a fixed set of legal values which are all required to be implemented by a conforming PEX server and cannot be inquired (for example, text path). If the PEX server finds a value outside the range of legal values, an *OutputCommand* error is reported, as described above.

A third category of enumerated types have a fixed set of legal values, but they are not all required to be implemented by a PEX server. Values supported by the PEX server can be inquired (for example, interior style). When the output command is rendered, values that are not supported by the PEX server are rendered with a default value and do not report errors.

If an output command contains a bitmask value, the PEX server must return an *OutputCommand* error if any undefined bits are set. Usually, these errors are not specifically mentioned in the output command descriptions below.

The client is not required to pass unit length normal vectors to the PEX server in output commands. The effect of rendering output primitives with normal vectors that are not unit length is implementation dependent.

3.3. Output Command Descriptions

The list below describes the format of the output commands that are supported. Each output command is a structure of type OUTPUT_COMMAND, which contains a 16-bit opcode that uniquely defines the output command (as well as uniquely identifying the structure element if the command is stored in a structure), a 16-bit size field which specifies the length of the output command in units of four bytes, and the data needed to specify the output command. The high-order bit of the opcode field is reserved to indicate whether the output command is one of the standard PEX output commands (high-order bit equals zero) or a non-standard or proprietary output command (high-order bit equals one).

Marker type

marker_type : MARKER_TYPE

When processed by a renderer, this command will modify the renderer's *marker_type* attribute. If the specified marker type is not supported by the PEX server, marker type 3 (*MarkerAsterisk*) is used.

Any integer value may be specified as the marker type in this output command.

Marker scale

scale : FLOAT

When processed by a renderer, this command will modify the renderer's *marker_scale* attribute. The specified scale is multiplied by the nominal marker size (see **PEXGetImpDepConstants**) and the result is mapped to the nearest marker size supported by the server.

Marker color index

color : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *marker_color* attribute, setting the color type to *Indexed* and the color value to the index specified by *color*. If the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Marker color

color : COLOR_SPECIFIER

When processed by a renderer, this command will modify the renderer's *marker_color* attribute, setting the color type and value as specified. If the color type is *Indexed* and the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Marker bundle index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *marker_bundle_index* attribute. If an undefined marker bundle index is specified by this output command, then default bundle index one is used. If table index one is not defined, the default values listed in Appendix D are used.

An *OutputCommand* error is reported if the bundle index in this output command is zero. |

Text font index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *text_font_index* attribute. The *text_font_index* selects which entry in the text font table (i.e., which font group) will be used to render text primitives. If an undefined text font index is specified by this output command, the default index one is used. If table index one is not defined, the default values listed in Appendix D are used.

An *OutputCommand* error is reported if the text font index in this output command is less than one.

Text precision

precision : TEXT_PRECISION

When processed by a renderer, this command will modify the renderer's *text_precision* attribute.

When a text or annotation text output primitive is interpreted, all of the ISTRING fragments in the text string are rendered in the same text precision. That is, if the font group selected by the current text font index consists of both X and PEX fonts, and if some of the ISTRING fragments in the string are rendered in X fonts, the text precision of the entire string must be dropped to at least *Char* precision.

If a *char_set* value is not available in the current font group, then the entire string is rendered using the default font group. If a *char_set* value is not available in the default font group, then that portion of the string is rendered in an implemenation dependent manner.

Character expansion

expansion : FLOAT

When processed by a renderer, this command will modify the renderer's *char_expansion* attribute. Only the magnitude of the specified expansion is considered. The specified expansion is compared to the minimum and maximum character expansion factors. These values depend on the font files that are in the font groups in the selected font table entry, which in turn depend on which X or PEX font files the client opened. For example, if the client opens all PEX font files (that is, all scalable and rotatable stroke fonts), then a continuous number of expansions are supported. If the expansion is smaller than the minimum character expansion factor, the minimum value is used. If the expansion is larger than the maximum character expansion factor, the maximum value is used.

Character spacing

spacing : FLOAT

When processed by a renderer, this command will modify the renderer's *char_spacing* attribute.

No errors or defaults are defined.

Text color index

color : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *text_color* attribute, setting the color type to *Indexed* and the color value to the index specified by *color*. If the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Text color

color : COLOR_SPECIFIER

When processed by a renderer, this command will modify the renderer's *text_color* attribute, setting the color type and value as specified. If the color type is *Indexed* and the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Character height

height : FLOAT

When processed by a renderer, this command will modify the renderer's *char_height* attribute. If the specified height or the computed width is not supported, the height or width is mapped to the nearest character height or width supported. These values depend on the font files that are in the font groups in the selected font table entry, which in turn depend on which X or PEX font files the client opened. For example, if the client opens all PEX font files (that is, all scalable and rotatable stroke fonts), then a continuous number of character sizes are supported.

Character up vector

up : VECTOR_2D

When processed by a renderer, this command will modify the renderer's *char_up_vector* attribute. If the character up vector is degenerate (it has a length of zero), the value <0, 1> is used.

Text path

path : TEXT_PATH

When processed by a renderer, this command will modify the renderer's *text_path* attribute.

An *OutputCommand* error is reported if the path is not *PathRight*, *PathLeft*, *PathUp*, or *PathDown*.

Text alignment

alignment : TEXT_ALIGNMENT

When processed by a renderer, this command will modify the renderer's *text_alignment* attribute.

An *OutputCommand* error is reported if the horizontal alignment is not *HalignNormal*, *HalignLeft*, *HalignCenter*, or *HalignRight*, or if the vertical alignment is not *ValignNormal*, *ValignTop*, *ValignCap*, *ValignHalf*, *ValignBase*, or *ValignBottom*.

Annotation text height

height : FLOAT

When processed by a renderer, this command will modify the renderer's *atext_height* attribute. If the specified height or the computed width is not supported, the height or width is mapped to the nearest annotation character height or width supported. These values depend on the font files that are in the font groups in the selected font table entry, which in turn depend on which X or PEX font files the client opened. For example, if the client opens all PEX font files (that is, all scalable and rotatable stroke fonts), then a continuous number of character sizes are supported.

Annotation text up vector

up : VECTOR_2D

When processed by a renderer, this command will modify the renderer's *atext_up_vector* attribute. If the annotation text up vector is degenerate (it has a length of zero), the value <0, 1> is used.

Annotation text path

path : TEXT_PATH

When processed by a renderer, this command will modify the renderer's *atext_path* attribute.

An *OutputCommand* error is reported if the path is not *PathRight*, *PathLeft*, *PathUp*, or *PathDown*.

Annotation text alignment

alignment : TEXT_ALIGNMENT

When processed by a renderer, this command will modify the renderer's *atext_alignment* attribute.

An *OutputCommand* error is reported if the horizontal alignment is not *HalignNormal*, *HalignLeft*, *HalignCenter*, or *HalignRight*, or if the vertical alignment is not *ValignNormal*, *ValignTop*, *ValignCap*, *ValignHalf*, *ValignBase*, or *ValignBottom*.

Annotation text style

index : ATEXT_STYLE

When processed by a renderer, this command will modify the renderer's *atext_style* attribute. If the specified style is not supported by the PEX server, annotation style 1 (*ATextNotConnected*) is used.

Any integer value may be specified as the style in this output command.

Text bundle index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *text_bundle_index* attribute. If an undefined text bundle index is specified by this output command, then default bundle index one is used. If table index one is not defined, the default values listed in Appendix D are used.

An *OutputCommand* error is reported if the bundle index in this output command is zero. |

Line type

line_type : LINE_TYPE

When processed by a renderer, this command will modify the renderer's *line_type* attribute. If the specified line type is not supported by the PEX server, line type 1 (*LineTypeSolid*) is used.

Any integer value may be specified as the line type in this output command.

Line width
width : FLOAT

When processed by a renderer, this command will modify the renderer's *line_width* attribute. The specified line width is multiplied by the nominal line width (see **PEXGetImpDepConstants**). The result is mapped to the nearest line width supported by the server.

Line color index
color : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *line_color* attribute, setting the color type to *Indexed* and the color value to the index specified by *color*. If the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Line color
color : COLOR_SPECIFIER

When processed by a renderer, this command will modify the renderer's *line_color* attribute, setting the color type and value as specified. If the color type is *Indexed* and the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Curve approximation
approx : CURVE_APPROX

When processed by a renderer, this command will modify the renderer's *curve_approx* attribute. If the specified method is not supported by the PEX server, method 1 (implementation dependent) is used.

Any integer value may be specified as the curve approximation method in this output command.

Polyline interpolation method
polyline_interp : POLYLINE_INTERP

When processed by a renderer, this command will modify the renderer's *polyline_interp* attribute. If the specified interpolation method is not supported by the PEX server, method 1 (*PolylineInterpNone*) is used.

Any integer value may be specified as the interpolation method in this output command.

Line bundle index
index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *line_bundle_index* attribute. If an undefined line bundle index is specified by this output command, then default bundle index one is used. If table index one is not defined, the default values listed in Appendix D are used.

An *OutputCommand* error is reported if the bundle index in this output command is zero. |

Surface interior style
interior_style : INTERIOR_STYLE

When processed by a renderer, this command will modify the renderer's *interior_style* attribute. If the specified style is not supported, style 1 (*InteriorStyleHollow*) is used.

An *OutputCommand* error is reported if the style is not *InteriorStyleHollow*, *InteriorStyleSolid*, *InteriorStylePattern*, *InteriorStyleHatch*, or *InteriorStyleEmpty*.

Surface interior style index
index : TYPE_OR_TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *interior_style_index* attribute. If the current *interior_style* is *InteriorStylePattern* or *InteriorStyleHatch*, the specified index is used to further define the rendering style of surface primitives. For *InteriorStylePattern*, if the specified pattern table index is not defined, table index one is used.† For *InteriorStyleHatch*, the index determines the hatch style and may be positive or negative. If the specified hatch style is not supported, style one is used. If hatch style one is not supported, the result is implementation dependent.

Surface color index

color : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *surface_color* attribute, setting the color type to *Indexed* and the color value to the index specified by *color*. If the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Surface color

color : COLOR_SPECIFIER

When processed by a renderer, this command will modify the renderer's *surface_color* attribute, setting the color type and value as specified. If the color type is *Indexed* and the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Surface reflection attributes

attr : REFLECTION_ATTR

When processed by a renderer, this command will modify the renderer's *reflection_attr* attribute.

No errors or defaults are defined.

Surface reflection model

reflection_model : REFLECTION_MODEL

When processed by a renderer, this command will modify the renderer's *reflection_model* attribute. If the specified reflection model is not supported by the PEX server, method 1 (*ReflectionNoShading*) is used.

Any integer value may be specified as the reflection model in this output command.

Surface interpolation method

surface_interp : SURFACE_INTERP

When processed by a renderer, this command will modify the renderer's *surface_interp* attribute. If the specified interpolation method is not supported by the PEX server, method 1 (*SurfaceInterpNone*) is used.

Any integer value may be specified as the interpolation method in this output command.

Backface surface interior style

interior_style : INTERIOR_STYLE

When processed by a renderer, this command will modify the renderer's *bf_interior_style* attribute. If the specified style is not supported, style 1 (*InteriorStyleHollow*) is used.

An *OutputCommand* error is reported if the style is not *InteriorStyleHollow*, *InteriorStyleSolid*, *InteriorStylePattern*, *InteriorStyleHatch*, or *InteriorStyleEmpty*.

Backface surface interior style index

index : TYPE_OR_TABLE_INDEX

† PHIGS requires that the interior style index must be greater than zero for *InteriorStylePattern*, but this is difficult for PEX to enforce, so a default action is defined instead.

When processed by a renderer, this command will modify the renderer's *bf_interior_style_index* attribute. If the current *bf_interior_style* is *InteriorStylePattern* or *InteriorStyleHatch*, the specified index is used to further define the rendering style of backfacing surface primitives. For *InteriorStylePattern*, if the specified pattern table index is not defined, table index one is used.‡ For *InteriorStyleHatch*, the index determines the hatch style and may be positive or negative. If the specified hatch style is not supported, style one is used. If hatch style one is not supported, the result is implementation dependent.

Backface surface color index

color : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *bf_surface_color* attribute, setting the color type to *Indexed* and the color value to the index specified by *color*. If the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Backface surface color

color : COLOR_SPECIFIER

When processed by a renderer, this command will modify the renderer's *bf_surface_color* attribute, setting the color type and value as specified. If the color type is *Indexed* and the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Backface surface reflection attributes

attr : REFLECTION_ATTR

When processed by a renderer, this command will modify the renderer's *bf_reflection_attr* attribute.

No errors or default are defined.

Backface surface reflection model

reflection_model : REFLECTION_MODEL

When processed by a renderer, this command will modify the renderer's *bf_reflection_model* attribute. If the specified reflection model is not supported by the PEX server, method 1 (*ReflectionNoShading*) is used.

Any integer value may be specified as the reflection model in this output command.

Backface surface interpolation method

surface_interp : SURFACE_INTERP

When processed by a renderer, this command will modify the renderer's *bf_surface_interp* attribute. If the specified interpolation method is not supported by the PEX server, method 1 (*SurfaceInterpNone*) is used.

Any integer value may be specified as the interpolation method in this output command.

Surface approximation

approx : SURFACE_APPROX

When processed by a renderer, this command will modify the renderer's *surface_approx* attribute. If the specified method is not supported by the PEX server, method 1 (implementation dependent) is used.

Any integer value may be specified as the surface approximation method in this output command.

Facet culling mode

culling_mode : CULL_MODE

‡ PHIGS requires that the backface interior style index must be greater than zero for *InteriorStylePattern*, but this is difficult for PEX to enforce, so a default action is defined instead.

When processed by a renderer, this command will modify the renderer's *culling_mode* attribute.

An *OutputCommand* error is reported if the mode is not *None*, *BackFaces*, or *FrontFaces*.

Facet distinguish flag

distinguish : BOOLEAN

When processed by a renderer, this command will modify the renderer's *distinguish* attribute.

An *OutputCommand* error is reported if the mode is not *True* or *False*.

Pattern size

size : VECTOR_2D

When processed by a renderer, this command will modify the renderer's *pattern_size* attribute. Only the magnitudes of the specified pattern size components are considered. If the current *interior_style* is *InteriorStylePattern*, the specified pattern size components are used. If either component is zero, the output command is ignored.

Pattern reference point

point : COORD_2D

When processed by a renderer, this command will modify the renderer's *pattern_ref_pt*, *pattern_ref_vec1*, and *pattern_ref_vec2* attributes. The *z* coordinate of the reference point will be set to zero, *pattern_ref_vec1* will be set to <1,0,0>, and *pattern_ref_vec2* will be set to <0,1,0>.

No errors or defaults are defined.

Pattern reference point and vectors

ref_pt : COORD_3D
vector1 : VECTOR_3D
vector2 : VECTOR_3D

When processed by a renderer, this command will modify the renderer's *pattern_ref_pt*, *pattern_ref_vec1*, and *pattern_ref_vec2* attributes. If the pattern vectors define a degenerate case (that is, if one of the vectors is zero length or if the vectors are parallel), the output command is ignored.

Interior bundle index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *interior_bundle_index* attribute. If an undefined interior bundle index is specified by this output command, then default bundle index one is used. If table index one is not defined, the default values listed in Appendix D are used.

An *OutputCommand* error is reported if the bundle index in this output command is zero.

Surface edge flag

onoff : SWITCH

When processed by a renderer, this command will modify the renderer's *surface_edges* attribute.

An *OutputCommand* error is reported if the flag is not *On* or *Off*.

Surface edge type

edge_type : SURFACE_EDGE_TYPE

When processed by a renderer, this command will modify the renderer's *surface_edge_type* attribute. If the specified edge type is not supported by the PEX server, edge type 1 (*SurfaceEdgeSolid*) is used.

Any integer value may be specified as the edge type in this output command.

Surface edge width

width : FLOAT

When processed by a renderer, this command will modify the renderer's *surface_edge_width* attribute. The specified edge width is multiplied by the nominal edge width (see **PEXGetImpDepConstants**). The result is mapped to the nearest edge width supported by the PEX server.

Surface edge color index

color : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *surface_edge_color* attribute, setting the color type to *Indexed* and the color value to the index specified by *color*. If the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Surface edge color

color : COLOR_SPECIFIER

When processed by a renderer, this command will modify the renderer's *surface_edge_color* attribute, setting the color type and value as specified. If the color type is *Indexed* and the specified color index is not defined, color index one is used. If color index one is not defined, the default values listed in Appendix D are used.

Edge bundle index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *edge_bundle_index* attribute. If an undefined edge bundle index is specified by this output command, then default bundle index one is used. If table index one is not defined, the default values listed in Appendix D are used.

An *OutputCommand* error is reported if the bundle index in this output command is zero.

Set individual ASF

attribute : ASF_ATTRIBUTE
source : ASF_VALUE

When processed by a renderer, this command will modify the specified ASF (aspect source flag) attribute in the renderer. Depending on the value of *attribute*, one of the following rendering pipeline attributes will be modified:

marker_type_asf
marker_scale_asf
marker_color_asf
text_font_index_asf
text_prec_asf
char_expansion_asf
char_spacing_asf
text_color_asf
line_type_asf
line_width_asf
line_color_asf
curve_approx_asf
polyline_interp_asf
interior_style_asf
interior_style_index_asf
surface_color_asf
surface_interp_asf
reflection_model_asf
reflection_attr_asf
bf_interior_style_asf
bf_interior_style_index_asf
bf_surface_color_asf
bf_surface_interp_asf
bf_reflection_model_asf
bf_reflection_attr_asf
surface_approx_asf
surface_edges_asf
surface_edge_type_asf
surface_edge_width_asf
surface_edge_color_asf

An *OutputCommand* error is reported if *attribute* does not refer to a valid ASF attribute or if *source* is not set to *Bundled* or *Individual*. |

Local transform 3D

comp_type : COMPOSITION
matrix : MATRIX

When processed by a renderer, this command will modify the renderer's *local_transform* attribute. If *comp_type* is *PreConcatenate*, the specified matrix is pre-concatenated to the local model transformation matrix. If *comp_type* is *PostConcatentate*, the specified matrix is post-concatenated to the local modeling transform. If *comp_type* is *Replace*, the specified matrix replaces the local modeling transform.

An *OuputCommand* error is reported if the composition type is not *PreConcatenate*, *PostConcatenate* or *Replace*.

Local transform 2D

comp_type : COMPOSITION
matrix : MATRIX_3X3

When processed by a renderer, this command will modify the renderer's *local_transform* attribute. This command is identical to *local transform 3D* except that the specified matrix is a 3×3 matrix instead of a 4×4 matrix. Before the concatenation occurs, the 3×3 matrix represented by

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix}$$

will be expanded to a 4×4 matrix as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \rightarrow \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{bmatrix}$$

An *OutputCommand* error is reported if the composition type is not *PreConcatenate*, *PostConcatenate* or *Replace*.

Global transform 3D

matrix : MATRIX

When processed by a renderer, this command will replace the renderer's *global_transform* attribute.

No errors or defaults are defined.

Global transform 2D

matrix : MATRIX_3X3

When processed by a renderer, this command will replace the renderer's *global_transform* attribute. This command is identical to "Global transform 3D" except that the specified matrix is a 3×3 matrix instead of a 4×4 matrix. Before the replacement occurs, the 3×3 matrix represented by

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix}$$

will be expanded to a 4×4 matrix as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & j \end{bmatrix} \rightarrow \begin{bmatrix} a & b & 0 & c \\ d & e & 0 & f \\ 0 & 0 & 1 & 0 \\ g & h & 0 & j \end{bmatrix}$$

No errors or defaults are defined.

Model clip
clip_switch : CLIP_INDICATOR

When processed by a renderer, this command will modify the renderer's *model_clip* attribute.

An *OutputCommand* error is reported if the model clip indicator is not *Clip* or *NoClip.*

Set model clip volume 3D
operator : OPERATOR
halfspaces : LISTofHALFSPACE

When processed by a renderer, this command will modify the renderer's *model_clip_volume* attribute. Each halfspace 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 halfspace, and the point is considered to be on the plane.

Each half-space is transformed by the current composite modeling transformation and combined with the current model clipping volume using the operater specified by *operator*. The model clip volume which results is not affected by subsequent changes to the composite modeling transformation. If the specified operator is not supported by the PEX server or if any halfspace is degenerate, the output command is ignored.

Set model clip volume 2D
operator : OPERATOR
halfspaces : LISTofHALFSPACE_2D

When processed by a renderer, this command will modify the renderer's *model_clip_volume* attribute. Each halfspace 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 halfspace, and the point is considered to be on the plane. The z component of each point and vector are assumed to be zero.

Each half-space is transformed by the current composite modeling transformation and combined with the current model clipping volume using the operater specified by *operator*. The model clip volume which results is not affected by subsequent changes to the composite modeling transformation. If the specified operator is not supported by the PEX server or if any halfspace is degenerate, the output command is ignored.

Restore model clip volume

When processed by a renderer, this command will modify the renderer's *model_clip_volume* attribute. The modeling clipping volume will be restored to its state as of the last structure invocation, or to the default state if no structure was invoked.

No errors or defaults are defined.

View index
index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *view_index* attribute. If the specified view index is not defined, index zero is used. If view index zero is not defined, the default values listed in Appendix D are used.

Light source state
enable : LISTofCARD16
disable : LISTofCARD16

When processed by a renderer, this command will modify the renderer'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.

An *OutputCommand* error is reported if a light in either the *enable* or *disable* list is zero, or if the same light is specified in both the lists. |

Depth cue index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *depth_cue_index* attribute. If the specified depth cue index is not defined, index zero is used. If depth cue index zero is not defined, the default values listed in Appendix D are used.

Pick ID

pickid : CARD32

When processed by a renderer, this command will modify the renderer's *pick_id* attribute.

No errors or defaults are defined.

HLHSR identifier

id : CARD32

When processed by a renderer, this command will modify the renderer's *HLHSR_identifier* attribute. This output command is a noop for all registered HLHSR modes except *ZBufferId*. If the renderer's *HLHSR_mode* is set to *ZBufferId* then a zero value will disable z-buffering and a value of one will enable it. For non-registered HLHSR modes the effect of this command is implementation dependent. If the specified HLHSR identifier is not supported by the PEX server, the HLHSR identifier used is implementation dependent. |

Color approximation index

index : TABLE_INDEX

When processed by a renderer, this command will modify the renderer's *color_approx_index* attribute. If the specified color approximation index is not defined, index zero is used. If index zero is not defined, the default values listed in Appendix D are used.

Rendering color model

model : COLOR_MODEL

When processed by a renderer, this command will modify the renderer's *rdr_color_model* attribute. If the specified color model is not supported by the PEX server, model 0 (implementation dependent) is used.

Any integer value may be specified as the color model in this output command.

Parametric surface characteristics

psc : PSURF_CHAR

When processed by a renderer, this command will modify the renderer's *psurf_char* attribute. If the specified type is not supported by the PEX server, type 1 (*None*) is used and the data record contents are implementation dependent.

Any integer value may be specified as the characteristics type in this output command. If an inconsistent value is specified in the data record, an *OutputCommand* error is reported. (For example, if the type is *IsoparametricCurves* and the curve placement is not *Uniform* or *NonUniform*, or if the type is *MC_LevelCurves* or *WC_LevelCurves* and the direction vector has zero length.)

Add names to name set

names : LISTofNAME

When processed by a renderer, this command will add names to the renderer's name set. If any name in the set of normal or inverted name set names is outside the supported range (see **PEXGetImpDepConstants**), that name is ignored.†

† PHIGS defines no errors for Add Names To Set.

Remove names from name set

names : LISTofNAME

When processed by a renderer, this command will remove names from the renderer's name set. If any name in the set of normal or inverted name set names is outside the supported range (see **PEXGetImpDepConstants**), that name is ignored.‡

Execute structure

s_id : STRUCTURE_ID

When processed by a renderer, this output command transfers flow-of-control to the specified structure. Processing of output commands will then commence with the first structure element in the structure specified by *s_id*. When all of the elements in the called structure have been processed, control will be returned. When this command is executed, all of the rendering pipeline attribute values in the renderer are saved. Then, the current global modeling transform is set to the current composite modeling transform and the current local modeling matrix is set to the identity matrix. When control is returned, the saved rendering pipeline attribute values are restored.

If *s_id* references a non-existent structure, an *OutputCommand* error is reported.* Note that it is not possible to have traversal time errors due to non-existent structures, since **PEXDestroyStructures** removes all references to structures it deletes.

Label

label : INT32

When processed by a renderer, this output command is a no-op. Its main usefulness is when used as a structure element to maintain the specified *label* as an aid to navigation during structure editing.

No errors or defaults are defined.

Application data

data : LISTofCARD8

When processed by a renderer, this output command is a no-op. Its main usefulness is when used as a structure element in order to maintain the specified client application data.

No errors or defaults are defined. Note that byte-swapping and floating point conversion are not done on this type of output command.

GSE

id : INT32
data : LISTofCARD8

When processed by a renderer, the effect of this command is implementation-dependent. Because of floating point and color format discrepancies across a network interface, it is not anticipated that the GSE will provide a useful extension mechanism, but it is provided for PHIGS compatibility purposes. If the specified GSE identifier is not supported by the PEX server, the output command is ignored. Note that byte-swapping and floating point conversions may not be done on this type of output command.

Marker 3D

points : LISTofCOORD_3D

When processed by a renderer, this command will cause marker primitives to be rendered. A marker is a geometric primitive with only one geometric attribute - its position. The list *points* contains a list of 3D coordinates, each of which specifies the position of a marker in modeling coordinates.

‡ PHIGS defines no errors for Remove Names From Set.
* PHIGS specifies that if the specified structure is non-existent, a new empty structure is created. In PEX, it is the client's responsibility to ensure that references to non-existent structure resource identifiers do not occur.

During the rendering process, the marker's position is transformed to a position in device coordinates. A marker has no geometric size, so geometric transformations will not affect the displayed size of the marker. The marker's color is transformed only by the depth-cueing computation (the light-source shading stage of the rendering pipeline affects only surfaces) and mapped to a device color. The clipping of markers whose position is inside the clipping volume but whose rendering is outside the clipping volume is implementation-dependent.

Depending on the setting of the marker attribute ASF values, the *marker_color*, *marker_type*, and *marker_scale* attributes are either obtained directly from the current marker attribute values or from the *marker_bundle_index*'th entry in the renderer's *marker_bundle*.

No errors or defaults are defined.

Marker 2D

points : LISTofCOORD_2D

When processed, this command will cause marker primitives to be drawn. This primitive functions exactly as the 3D marker primitive except that modeling coordinate positions are specified using only *x*- and *y*-coordinates, and the *z*-coordinate is always assumed to be zero.

No errors or defaults are defined.

Text 3D

origin : COORD_3D
vector1 : VECTOR_3D
vector2 : VECTOR_3D
string : ISTRING

When processed by a renderer, this command will cause a text string to be rendered. The parameter *string* contains the text string to be rendered. A string is a list of data records, each of which contains *char_set*, *char_set_width* and *encoding_state* fields, as well as a list of the characters that actually make up that portion of the string. The *char_set* field is an index into the current font group. Font groups have individual fonts numbered starting at one, so a value of three would select the third font in the font group currently being used. If a *char_set* value is not available in the current font group, then the entire string will be rendered using the default font group. If a *char_set* value is not available in the default font group, then that portion of the string will be rendered in an implementation-dependent manner. The *char_set_width* indicates whether characters in the string are 8-, 16-, or 32-bit values. Characters in the string 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 string is located on a plane defined by its position and direction vectors. The origin of the string is a point in modeling coordinates indicated by *origin*, and the string's direction is indicated by the direction vectors *vector1* and *vector2*. *Vector1* defines the positive *x*-axis of the text local coordinate system. *Vector2* defines the positive *y*-axis of the text local coordinate system.

Depending on the setting of the text attribute ASF values, the *text_color*, *text_precision*, *char_expansion*, *char_spacing*, and *text_font_index* attributes are either obtained directly from the current text attribute values or from the *text_bundle_index*'th entry in the renderer's *text_bundle*. The *char_height*, *char_up_vector*, *text_path*, and *text_alignment* attributes are also used when drawing the text primitive. An attempt is made to render the text string as accurately as possible with the font group named by the current *text_font_index*. The directions specified by *char_up_vector* and *text_path* will be relative to the text local coordinate system.

During the rendering process, a string's position is transformed to a position in device coordinates. The string's color is transformed only by the depth-cueing computation (the light-source shading stage of the rendering pipeline affects only surfaces) and mapped to a device color. The text string is clipped depending on the current text precision value. If the text precision is *String*, clipping is done in an implementation-dependent fashion. If the text 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.

If either of the text direction vectors is zero length or if the vectors are parallel, the vectors <1,0,0> and <0,1,0> are used.

Text 2D

origin : COORD_2D
string : ISTRING

When processed, this command will cause a text primitive to be drawn. This primitive functions exactly as the 3D text primitive except that it has no direction vectors, the modeling coordinate position is specified using only *x*- and *y*- coordinates, and the *z*-coordinate is always assumed to be zero.

No errors or defaults are defined.

Annotation text 3D

origin : COORD_3D
offset : COORD_3D
string : ISTRING

When processed by a renderer, this command will cause an annotation text string to be rendered. The parameter *string* contains the text string to be rendered. A string is a list of data records, each of which contains *char_set*, *char_set_width* and *encoding_state* fields, as well as a list of the characters that actually make up that portion of the string. The *char_set* field is an index into the current font group. Font groups have individual fonts numbered starting at one, so a value of three would select the third font in the font group currently being used. If a *char_set* value is not available in the current font group, then the entire string will be rendered using the default font group. If a *char_set* value is not available in the default font group, then that portion of the string will be rendered in an implementation-dependent manner. The *char_set_width* indicates whether characters in the string are 8-, 16-, or 32-bit values. Characters in the string 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 origin of the string is a point in modeling coordinates indicated by *origin*. An offset value in normalized projection coordinates is specified by *offset*. The point at which the annotation text string is to be placed is called the annotation point, and is computed by adding *offset* to the transformed *origin* point. The *z*-component of the annotation point specifies the the *x-y* plane in normalized projection coordinate space on which the annotation text string will be placed.

Depending on the setting of the text attribute ASF values, the *text_color*, *text_precision*, *char_expansion*, *char_spacing*, and *text_font_index* attributes are either obtained directly from the current text attribute values or from the *text_bundle_index*'th entry in the renderer's *text_bundle*. The *atext_height*, *atext_path*, *atext_alignment*, *atext_up_vector*, and *atext_style* attributes are also used when rendering the text string.

The annotation text string's color is transformed only by the depth-cueing computation (the light-source shading stage of the rendering pipeline affects only surfaces) and mapped to a device color. The entire annotation text primitive is clipped if *origin* is clipped. If *origin* is not clipped by modeling, view, or workstation clipping, the annotation text will be clipped according to text clipping rules and the connection line, if present, will be clipped according to polyline clipping rules, except that modeling clipping will be ignored. The current set of polyline attributes will be used to draw the connection line if it is to be drawn.

No errors or defaults are defined.

Annotation text 2D

origin : COORD_2D
offset : COORD_2D
string : ISTRING

When processed, this command will cause an annotation text primitive to be drawn. This primitive functions exactly as the 3D annotation text primitive except that origin and offset positions are specified using only *x*- and *y*- coordinates, and the *z*-coordinate is always assumed to be zero.

No errors or defaults are defined.

Polyline 3D

vertices : LISTofCOORD_3D

When processed by a renderer, this command will cause a polyline to be rendered. The polyline is defined by the list of vertices that are specified in *vertices*, each of which is a coordinate in the modeling coordinate system. The vertices are joined together by line segments. The first vertex of a polyline is connected to the second, the second connected to the third, and so on. The last vertex is *not* connected to the first.

Depending on the setting of the line attribute ASF values, *line_color*, *line_type*, and *line_width* attributes are either obtained directly from the current line attribute values or from the *line_bundle_index*'th entry in the renderer's *line_bundle*.

Polylines have no geometric width, only length, so transformations will affect only the displayed length of a polyline. The polyline colors are transformed only by the depth-cueing computation (the light-source shading stage of the rendering pipeline affects only surfaces) and mapped to device colors. Polylines are not displayed if they are outside the currently defined clipping volume. Polylines that cross the clipping volume are clipped, and only the portion(s) inside the clipping volume is (are) displayed.

At structure creation time, if the polyline has fewer than two vertices, it is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

Polyline 2D

vertices : LISTofCOORD_2D

When processed, this command will cause a polyline primitive to be drawn. This primitive functions exactly as the 3D polyline primitive except that modeling coordinate positions are specified using only *x*- and *y*-coordinates, and the *z*-coordinate is always assumed to be zero.

At structure creation time, if the polyline has fewer than two vertices, it is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

Polyline set 3D with data

color_type : COLOR_TYPE
vert_attributes : BITMASK_SHORT |
vertices : LISTofLISTofVERTEX

When processed by a renderer, this command will cause a series of polylines to be rendered. The behavior of this primitive is identical to that of the 3D polyline primitive, except that multiple polylines can be drawn, and additional information can be specified at each polyline vertex. Color values that are passed will be of the type specified by *color_type*. The parameter *vert_attributes* indicates the attributes which are specified at each polyline vertex. The components of the vertex attributes bitmask are, in order:

color COLOR

If any of the attribute bits is set, the corresponding attributes must be present for each vertex, and they must be passed after the coordinate data for each vertex in the order that they appear in the list above.

If color values are passed per vertex, they are considered to be part of the primitive and are used instead of the *line_color* attribute. In addition, the use of per-vertex colors is affected by the *polyline_interp* attribute, which is obtained directly from the *polyline_interp* value if the *polyline_interp_asf* attribute is set to *Individual* or from the *line_bundle_index*'th entry in the renderer's *line_bundle*. The *polyline_interp* attribute defines how color values between the vertices are to be computed.

At structure creation time, if any polyline in the set has fewer than two vertices, it is stored in the structure, but when this output command is interpreted, the polyline with insufficient data has no visual effect. In immediate mode, such a polyline is ignored.

Non-uniform B-spline curve

order : CARD16
type : COORD_TYPE
tmin : FLOAT
tmax : FLOAT
knots : LISTofFLOAT
points : LISTofCOORD

When processed by a renderer, this command will cause a non-uniform B-spline curve to be rendered. The *order* is specified as a positive integer. The spline shape is specified using a list of knots in the parametric coordinate space, plus a list of control points that are specified in modeling coordinates. In general, the number of control points must be at least as large as the order. The number of control points plus the spline order must equal the number of knots. The *knots* sequence must form a non-decreasing sequence of numbers.

The *type* parameter specifies whether the curve is *Rational* or *NonRational*. If the *type* is *Rational*, then the point list must be provided as homogeneous modeling coordinates (COORD_4D), otherwise the the point list must be 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.

Depending on the setting of the line attribute ASF values, *line_color*, *line_type*, *line_width*, and *curve_approx* attributes are either obtained directly from the current line attribute values or from the *line_bundle_index*'th entry in the renderer's *line_bundle*.

When an output command of this type is interpreted, if the curve order is not supported by the PEX server, the output command has no visual effect. In immediate mode, such a primitive is ignored.

Several conditions may cause an *OutputCommand* error to be reported: *type* is not *Rational* or *NonRational*, *order* is less than one, 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, *tmin* and *tmax* are inconsistent with the knots, or a *Rational* control point has a w coordinate that is less than or equal to zero.

Fill area 3D

shape : SHAPE
ignore_edges : BOOLEAN
vertices : LISTofCOORD_3D

When processed by a renderer, this command will cause a fill area primitive to be rendered. A fill area is defined by a list of vertices which are to be joined together to form a planar surface. (Fill areas are not strictly required to be planar, but strange shading artifacts can occur if a fill area is not planar or nearly so.) The first vertex of a fill area is connected to the second, the second connected to the third, and so on. The last vertex is implicitly connected to the first.

During the rendering process, the fill area vertices are transformed to positions in device coordinates. The surface colors are transformed by the light source shading computation (which utilizes the interior style and the reflection model) and are further modified by the depth-cueing computation and mapped to device colors. Fill areas are not displayed if they are outside the currently-defined clipping volume. Fill areas that cross the clipping volume are clipped, and only the portion(s) inside the clipping volume is (are) displayed.

A fill area can cross over itself to create a complex shape. The odd-even rule is used for determining the points that lie in the interior of a fill area. The *shape* parameter is passed as a hint as to the type of fill area that is defined by the *vertices*. A shape hint of *Unknown* means that nothing is known about the shape of the fill area. A shape of *Complex* means that the fill area edges may self-intersect. A shape of *Nonconvex* means that the edges do not self-intersect, but the fill area is not wholly convex. *Convex* means that all of the interior angles of the fill area are convex. Fill areas that are of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner. PEX server extensions may ignore the shape hint and treat all fill area primitives as shape *Unknown*. The *ignore_edges* parameter specifies whether or not surface edge attributes are to be applied to the fill area primitives. If it is *True*, no surface edges will ever be drawn for the *fill area*. If *False*, surface edges will be drawn using the current surface edge attributes if the surface edge flag is *On*. Depending on the setting of the surface edge attribute ASF values, the *surface_edges*, *surface_edge_color*, *surface_edge_type*, and *surface_edge_width* attributes are either obtained directly from the current surface edge attribute values or from the *edge_bundle_index*'th entry in the renderer's *edge_bundle*.

Depending on the setting of the surface attribute ASF values, the *surface_color*, *interior_style*, *interior_style_index*, *surface_interp*, and *reflection_model* attributes are either obtained directly from the current surface attribute values or from the *interior_bundle_index*'th entry in the renderer's *interior_bundle*. If, when rendered, the fill area is determined to be front-facing with respect to the point of view, the *surface_color* (obtained as previously described) and *reflection_attr* attributes are used to compute the color(s) of the fill area. If the fill area is determined to be back-facing with respect to the point of view, the *bf_surface_color* and *bf_reflection_attr* attributes are used instead. Regardless of the orientation of the fill area, if the *interior_style* is *Pattern*, the *pattern_size, pattern_ref_pt*, *pattern_ref_vec1*, and *pattern_ref_vec2* attributes may be used to pattern the fill area.

At structure creation time, if the fill area has fewer than three vertices, it is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

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

Fill area 2D

shape : SHAPE
ignore_edges : BOOLEAN
vertices : LISTofCOORD_2D

When processed, this command will cause a fill area primitive to be drawn. This primitive functions exactly as the 3D fill area primitive except that modeling coordinate positions are specified using only *x*- and *y*-coordinates, and the *z*-coordinate is always assumed to be zero.

At structure creation time, if the fill area has fewer than three vertices, it is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

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

Fill area 3D with data

shape : SHAPE
ignore_edges : BOOLEAN
color_type : COLOR_TYPE
facet_attributes : BITMASK_SHORT |
vert_attributes : BITMASK_SHORT |
facet : FACET

When processed by a renderer, this command will cause a fill area to be rendered. The behavior of this primitive is identical to that of the 3D fill area primitive, except that additional information can be specified for the fill area itself and for each vertex. Color values that are passed will be of the type specified by *color_type*.

The parameter *facet_attributes* indicates the attributes which are specified for the fill area. The components of the facet attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present in the data that defines the fill area facet and they must be passed in the order that they appear in the list above. If a color value is passed per facet, it is taken to be the intrinsic color of the front face of the facet. If a normal is passed per facet, it is taken to be the normal to the facet. (Normals are assumed to be unit vectors. The effect of rendering with normals that are not unit vectors is implementation dependent.) |

The parameter *vert_attributes* specifies the attributes which are specified at each fill area vertex. The components of the vertex attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attributes must be present for each vertex, and they must be passed after the coordinate data for each vertex in the order that they appear in the list above.

If color values are passed per vertex, they are considered to be part of the primitive and are used instead of the *surface_color* attribute. Vertex colors will be utilized rather than facet colors if both are provided. If normals are passed per vertex, they are taken to be the normals at the vertices of the fill area. In addition, the use of per-vertex colors is affected by the *surface_interp* attribute, which is obtained directly from the *surface_interp* value if the *surface_interp_asf* attribute is set to *Individual* or from the *interior_bundle_index*'th entry in the renderer's *interior_bundle*. The *surface_interp* attribute defines how color values between the vertices are to be computed.

The *shape* parameter is passed as a hint as to the type of fill area that is defined by the vertices. A shape hint of *Unknown* means that nothing is known about the shape of the fill area. A shape of *Complex* means that the fill area edges may self-intersect. A shape of *Nonconvex* means that the edges do not self-intersect, but the fill area is not wholly convex. *Convex* means that all of the interior angles of the fill area are convex. Fill areas that are of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner. PEX server extensions may ignore the shape hint and treat all fill area primitives as shape *Unknown*. The *ignore_edges* parameter specifies whether or not surface edge attributes are to be applied to the fill area primitives. If it is *True*, no surface edges will ever be drawn for the *fill area*. If *False*, surface edges will be drawn using the current surface edge attributes if the surface edge flag is *On*.

At structure creation time, if the fill area has fewer than three vertices, it is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

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

Fill area set 3D

shape : SHAPE
ignore_edges : BOOLEAN
contour_hint : CONTOUR
vertices : LISTofLISTofCOORD_3D

When processed by a renderer, this command will cause a fill area set primitive to be drawn. This primitive is essentially a set of fill area primitives which, together, define a polygon with islands or holes. The planarity of the resulting polygon is not verified, but the results of processing a non-planar fill area set are implementation dependent. The *ignore_edges* parameter will be applied to all of the fill areas in the fill area set. |
|
|

The *shape* parameter is passed as a hint as to the type of contours that comprise the fill area set. A shape hint of *Unknown* means that nothing is known about the shape of the constituent contours. A shape of *Complex* means that some contours of the fill area set may have edges that self-intersect. A shape of *Nonconvex* means that none of the contours of the fill area set have edges that self-intersect, but some may not be wholly convex. *Convex* means that all of the interior angles of all of the contours are convex. Contours that are of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner. PEX server extensions may ignore the shape hint and treat all constituent contours as shape *Unknown*.

The *contour_hint* parameter provides further information as to the relationships between contours in the fill area set. If *contour_hint* is *Disjoint*, all contours will be spatially disjoint. No overlapping or intersection occurs between any contours in the fill area set. If *contour_hint* is *Nested*, contours will either be disjoint or wholly contained within another contour. No contour will have edges that intersect or are coincident with edges of any other contour. If *contour_hint* is *Intersecting*, separate contours may have edges that are coincident or overlap. If *contour_hint* is *Unknown*, nothing is known about the interrelationships between contours. Fill area sets with contours that have higher complexity interrelationships than that indicated by their contour hint are drawn in an implementation-dependent manner. PEX server extensions may ignore the contour hint and treat all fill area sets as *Unknown*.

At structure creation time, if any of the contours of the fill area set has fewer than three vertices, or if there are no contours defined, the output command is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

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

Fill area set 2D

shape : SHAPE
ignore_edges : BOOLEAN
contour_hint : CONTOUR
vertices : LISTofLISTofCOORD_2D

When processed, this command will cause a fill area set primitive to be drawn. This primitive functions exactly as the 3D fill area set primitive except that modeling coordinate positions are specified using only *x*- and *y*- coordinates, and the *z*-coordinate is always assumed to be zero.

At structure creation time, if any of the contours of the fill area set has fewer than three vertices, or if there are no contours defined, the output command is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

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

Fill area set 3D with data

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

When processed by a renderer, this command will cause a fill area set to be rendered. The behavior of this primitive is identical to that of the 3D fill area set primitive, except that additional information can be specified for each of the fill area sets, for each edge, and for each vertex. Color values that are passed will be | of the type specified by *color_type*.

The parameter *facet_attributes* indicates the attributes which are specified in the *facet_data* parameter. The components of the facet attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attributes must be present in the *facet_data* parameter and must be passed in the order that they appear in the list above. If a color value is passed as part of *facet_data*, it is taken to be the intrinsic color of the front face of the fill area set.

The parameter *vert_attributes* specifies the attributes which are specified at each fill area set vertex. The components of the vertex attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D
edges	SWITCH

If any of the attribute bits is set, the corresponding attributes must be present for each vertex, and they must be passed after the coordinate data for each vertex in the order that they appear in the list above.

If color values are passed as part of *facet_data* or as part of the *vertices* list, they are considered to be part of the primitive and are used instead of the *surface_color* attribute. Vertex colors will be utilized rather than facet colors if both are provided. If normals are passed per vertex, they are taken to be the normals at the vertices of the fill area. If surface edge flags are specified per vertex, each flag specifies whether to draw the edge from the vertex with which the flag is specified to the next vertex. (E.g., for a facet with four vertices, the edge flag associated with vertex #1 indicates whether to draw edge #1-#2, edge flag #2 specifies edge #2-#3, edge flag #3 specifies edge #3-#4, and edge flag #4 specifies edge #4-#1.) Surface edges are always drawn with the surface edge color, never with per facet or per vertex colors.

In addition, the use of per-vertex colors is affected by the *surface_interp* attribute, which is obtained directly from the *surface_interp* value if the *surface_interp_asf* attribute is set to *Individual* or from the *interior_bundle_index*'th entry in the renderer's *interior_bundle*. The *surface_interp* attribute defines how color values between the vertices are to be computed.

At structure creation time, if any of the contours of the fill area set has fewer than three vertices, or if there are no contours defined, the output command is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

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

Triangle strip

color_type : COLOR_TYPE
facet_attributes : BITMASK_SHORT |
vert_attributes : BITMASK_SHORT |
facet_data : LISTofOPT_DATA
vertices : LISTofVERTEX

When processed by a renderer, this command will cause a triangle strip primitive to be drawn. Color values that are passed will be of the type specified by *color_type*. The parameter *facet_attributes* indicates the attributes which are specified for each facet of the triangle strip. The components of the facet attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present in *facet_data*, which is the data that defines each triangular facet. The attributes that are passed in this way must be passed in the order that they appear in the list above. If a color value is passed per facet, it is taken to be the intrinsic color of the front face of the facet. If a normal is passed per facet, it is taken to be the normal to the facet. (Normals are assumed to be unit vectors. The effect of rendering with normals that are not unit vectors is implementation dependent.) There will be n-2 entries in the *facet_data* list, where n is the number of entries in the *vertices* list. |

The parameter *vert_attributes* specifies the attributes which are specified at each triangle strip vertex. The components of the vertex attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present for each vertex, and it must be passed after the coordinate data for each vertex. The attributes that are passed in this way must be passed in the order that they appear in the list above.

If color values are passed per vertex, they are considered to be part of the primitive and are used instead of the *surface_color* attribute. Vertex colors will be utilized rather than facet colors if both are provided. If normals are passed per vertex, they are taken to be the normals at the vertices of the facet.

The triangle strip is created from the vertex array. The strip is composed of n-2 triangles, where n is the number of vertices. The first triangle is formed from the first three vertices in the list, the second triangle is formed by the second through the fourth vertices in the list, etc., up to the last triangle, which is formed by the last three vertices in the list.

All attributes that affect the representation of fill area sets also affect the representation of the triangle strip primitive.

At structure creation time, if the triangle strip has fewer than three vertices, the output command is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

Quadrilateral mesh

shape : SHAPE
color_type : COLOR_TYPE
m_pts : CARD16
n_pts : CARD16
facet_attributes : BITMASK_SHORT |
vert_attributes : BITMASK_SHORT |
facet_data : LISTofOPT_DATA
vertices : LISTofVERTEX

When processed by a renderer, this command will cause a quadrilateral mesh primitive to be rendered. The *shape* parameter is passed as a hint as to the type of quadrilaterals that comprise the primitive. A shape hint of *Unknown* means that nothing is known about the shape of the constituent quadrilaterals. A shape of *Complex* means that the 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 higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner. Color values that are passed will be of the type specified by *color_type*.

The parameter *facet_attributes* indicates the attributes which are specified for each facet of the quadrilateral mesh. The components of the facet attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present in *facet_data*, which is the data that defines each quadrilateral facet. The attributes that are passed in this way must be passed in the order that they appear in the list above. If a color value is passed per facet, it is taken to be the intrinsic color of the front face of the facet. If a normal is passed per facet, it is taken to be the normal to the facet. (Normals are | assumed to be unit vectors. The effect of rendering with normals that are not unit vectors is implementation | dependent.)

The parameter *vert_attributes* specifies the attributes which are specified at each quadrilateral mesh vertex. The components of the vertex attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present for each vertex, and it must be passed after the coordinate data for each vertex. The attributes that are passed in this way must be passed in the order that they appear in the list above.

If color values are passed per vertex, they are considered to be part of the primitive and are used instead of the *surface_color* attribute. Vertex colors will be utilized rather than facet colors if both are provided. If normals are passed per vertex, they are taken to be the normals at the vertices of the facet.

The surface will be created from a vertex array that is stored in row major order (i.e., the column number varies fastest as vertices are stored in the array). The (ith,jth), (i+1th,jth), (i+1th,j+1th) and (ith,j+1th) vertices are connected to create a single facet. Adjacent vertices are interconnected until the entire facet network is processed. There are *m_pts* × *n_pts* entries in the *vertices* array, and there are *(m_pts-1)* × *(n_pts-1)* entries in the *facet_data* array if any per-facet attributes are passed. *m_pts* is the number of columns in the vertex array and *n_pts* is the number of rows. |

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.

All attributes that affect the representation of fill area sets also affect the representation the quadrilateral mesh primitive.

At structure creation time, if either *m_pts* or *n_pts* is less than two, the output command is stored in the structure, but when this output command is interpreted, it has no visual effect. In immediate mode, such a primitive is ignored.

An *OutputCommand* error is reported if the *shape* parameter is invalid.

Set of fill area sets

shape : SHAPE
color_type : COLOR_TYPE
fas_attributes : BITMASK_SHORT |
vert_attributes : BITMASK_SHORT |
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

When processed by a renderer, this command will draw a set of fill area sets that may be connected (i.e., individual fill area sets may share geometry and attribute information at vertices). Shading calculations and transformations need only be performed once per shared vertex instead of once for every fill area set that shares the vertex. Similarly, data can be transmitted across the network once per unique vertex instead of once for every fill area set sharing the vertex.

The *shape* parameter is passed as a hint as to the type of contours that comprise each of the fill area sets. A shape hint of *Unknown* means that nothing is known about the shape of the constituent contours. A shape of *Complex* means that some contours of the fill area sets may have edges that self-intersect. A shape of *Nonconvex* means that none of the contours of the fill area sets have edges that self-intersect, but some may not be wholly convex. *Convex* means that all of the interior angles of all of the contours are convex. (Note that a fill area set with more than one contour is always allowed to have contours that intersect. It is quite possible that the only times that the fastest rendering code path can be taken are if the number of contours in a fill area set is equal to one or if *contours_all_1* is *True*, and the shape flag for the set of fill area sets primitive is *Convex*.) Contours that are of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner. PEX server extensions may ignore the shape hint and treat all constituent contours as shape *Unknown*.

The *contour_hint* parameter provides further information as to the relationships between contours in each of the fill area sets. If *contour_hint* is *Disjoint*, all contours will be spatially disjoint. No overlapping or intersection occurs between any contours in any of the fill area sets. If *contour_hint* is *Nested*, contours will either be disjoint or wholly contained within another contour. No contour will have edges that intersect or are coincident with edges of any other contour. If *contour_hint* is *Intersecting*, separate contours may have edges that are coincident or overlap. If *contour_hint* is *Unknown*, nothing is known about the interrelationships between contours. Fill area sets with contours that have higher complexity interrelationships than that indicated by their contour hint are drawn in an implementation-dependent manner. PEX server extensions may ignore the contour hint and treat all fill area sets as *Unknown*.

Color values that are passed will be of the type specified by *color_type*.

The parameter *fas_attributes* indicates the attributes that are specified for each fill area set. The components of the *fas_ attributes* bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present in *per_fas_data*, which contains one data record for each fill area set. The attributes that are passed in this way must be passed in the order that they appear in the list above. If a color value is passed, it is taken to be the intrinsic color of the front face of the fill area set. If a normal is passed, it is taken to be the normal to the fill area set. (Normals are assumed to be unit vectors. The effect of rendering with normals that are not unit vectors is implementation dependent.) If *fas_attributes* is null, the *per_fas_data* list will be null as well. Otherwise, there will be *num_fas* entries in *per_facet_data*.

The parameter *vert_attributes* indicates the attributes that are provided at each vertex in the list of unique vertices specified by *per_vert_data*. The components of the vertex attributes bitmask are, in order:

color	COLOR
normal	VECTOR_3D

If any of the attribute bits is set, the corresponding attribute must be present for each vertex in the *per_vert_data* list, and it must be passed after the coordinate data for each vertex. The attributes that are passed in this way must be passed in the order that they appear in the list above. If color values are passed per vertex, they are considered to be part of the primitive and are used instead of the *surface_color* attribute. If normals are passed per vertex, they are taken to be the normal at the indicated vertex, and will be used by all contours that share the vertex. There will always be *num_verts* entries in the *per_vert_data* list.

The parameter *edge_attributes* specifies the attributes that are specified at each edge. The components of the edge attributes bitmask are, in order:

edges	SWITCH

If any of the attribute bits is set, the corresponding attribute must be present for each edge in the list *per_edge_data*. If none of the attribute bits are set, the list *per_edge_data* will be an empty list. Otherwise, it will contain *num_edges* entries, each of which contains a switch indicating whether or not the corresponding edge should be drawn. This list is a flattened list without counts, so if it is nonempty, it is up to the PEX server extension to maintain a pointer to the proper position in this list while processing the data in *contours*. If edge switches are supplied, each flag specifies whether to draw the edge from the vertex with which the flag is specified to the next vertex. (E.g., for a contour with four vertices, the edge flag associated with vertex #1 indicates whether to draw edge #1-#2, edge flag #2 specifies edge #2-#3, edge flag #3 specifies edge #3-#4, and edge flag #4 specifies edge #4-#1.) Surface edges are always drawn with the surface edge color, never with per-fill-area-set or per-vertex colors.

The connectivity of the primitive is defined by the *contours* array. The number of contours in the first fill area set is contained as the first entry in the *contours* array. The number of contours is followed by a list of data records, one for each of the contours in the fill area set. If this number is *n*, then the next *n* data records in the *contours* array are used to define the first fill area set. The value following the contour count contains the number of vertices in the first contour of the first fill area set. If this number is *m*, then the next *m* values in the array comprise the data record for the first contour. This data record contains the indices for the vertices of the first contour. Depending on *n*, the next value in the list is either the number of vertices in the second contour for the first fill area set, or it is the number of contours in the second fill area set. As a special case, if *contours_all_1* is *True*, then the contour count field in each fill area set is guaranteed to be one.

Vertices are numbered with indices starting from zero (i.e., the first vertex is referenced as vertex 0).

All attributes that affect the representation of fill area sets also affect the representation of the set of fill area sets primitive.

An *OutputCommand* error is reported if any of the edge flags in the *per_edge_data* array 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 there are vertex indices in the lists in the contours array.

Non-uniform B-spline surface

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

When processed by a renderer, this command will draw a non-uniform B-spline surface. It generates the spline surface as a function of the parametric variables u and v. *U_order* and *v_order* indicate the order of the parametric variables and are specified as positive integers. The spline shape is specified using two lists of knots in the parametric coordinate space, plus an array of control points that are specified in modeling coordinates. The *u_knots* sequence and the *v_knots* sequence must each form a non-decreasing sequence of numbers. *Mpts* indicates the number of points in the *u* direction and *npts* indicates the number of points in the *v* direction. The points are stored in the vertex array in row major order (i.e., the column number varies fastest as vertices are stored in the array) and the rows increase in the direction of increasing *v*.

The minimum and maximum knot values in *u_knots* define the range over which the B-spline surface is evaluated in the *u* parametric direction, and the minimum and maximum knot values in *v_knots* define the range over which the B-spline surface is evaluated in the *v* parametric direction.

The *type* parameter specifies whether the surface is *Rational* or *NonRational*. If the *surface_type* is *Rational*, then the point list must be provided as homogeneous modeling coordinates (COORD_4D), otherwise the point list must be provided as non-homogeneous modeling coordinates (COORD_3D).

In addition to the parametric bounds, a list of trimming loops may be specified. Trimming loops serve to further restrict the region in parametric coordinate space over which the B-spline surface is to be evaluated. Each trimming loop is defined as a list of one or more B-spline trimming curves that are connected head-to-tail. Each trim curve can be *Rational* or *NonRational*, have a different order, etc. The list must be explicitly closed, so that the tail of the last B-spline curve joins the head of the first B-spline curve in each trimming loop. Each trimming curve is parameterized independently. If there is a floating point inaccuracy in closure or in head-to-tail connectivity between B-spline curves , closure or connectivity will be assumed. B-spline curves for trimming loops are defined in the parameter space of the surface and may not go outside the parameter space of the surface. When no trimming loops are specified, the rectangular parameter limits of the surface are rendered as the edges of the surface based on the edge flag attribute.

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 will cause the interior of the loop to be retained and the exterior to be clipped away. A clockwise loop will cause 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 B-spline curve may intersect itself and no trimming loop may intersect itself or any other trimming loop. Trimming loops that do not obey these rules will result in an implementation-dependent rendering. |

Each B-spline curve has a visibility flag that controls its visibility for the purposes of surface edge display. Depending on the settings of a renderer's surface edge attributes and the visibility flags associated with trim curves, the B-spline curves in trimming loops may be drawn as surface edges. |

All attributes that affect the representation of fill area sets also affect the representation of the non-uniform B-spline surface primitive. In addition, the *surface_approx* attribute is used to determine how to approximate the B-spline surface and the *psurf_char* attribute is used to specify the appearance of the B-spline surface.

If either of the surface orders or any of the trim curve orders is not supported by the PEX server, the output command has no visual effect. In immediate mode, such a primitive is ignored.

Several conditions may cause an *OutputCommand* error to be reported: *type* is not *Rational* or *NonRational*, *order* is less than one, there are fewer control points in either the u or v direction than the order, the order is inconsistent with the number of knots and control points, the knots are not non-decreasing, a *Rational* control point has a w coordinate that is less than or equal to zero, a trim curve order is less than 2, a trim curve does not have enough control points for its order, a trim curve's order is inconsistent with its number of knots and control points, a trim curve's knot sequence is not non-decreasing, or a trim curve's parameter range is inconsistent with its knots.

Cell array 3D

point1 : COORD_3D
point2 : COORD_3D
point3 : COORD_3D
dx : CARD32
dy : CARD32
colors : LISTofTABLE_INDEX

When processed by a renderer, this command will cause a cell array primitive to be rendered. Color values that are passed will be of type *Indexed*. A cell array is a parallelogram of equal-sized cells, each of which is a parallelogram and has a single color. Each cell has a width defined by:

$$width = \frac{\sqrt{(point1.x - point2.x)^2 + (point1.y - point2.y)^2 + (point1.z - point2.z)^2}}{dx}$$

and a height defined by:

$$height = \frac{\sqrt{(point1.x - point3.x)^2 + (point1.y - point3.y)^2 + (point1.z - point3.z)^2}}{dy}$$

The colors are specified in a one-dimensional array of size $dx \times dy$. The color of each cell is specified by the index of the corresponding element in the *colors* array. 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, color index one is used. If color index one is not defined, the default values in Appendix D are used.

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

Cell array 2D

point1 : COORD_2D
point2 : COORD_2D
dx : CARD32
dy : CARD32
colors : LISTofTABLE_INDEX

When processed, this command will cause a cell array primitive to be drawn. This primitive functions exactly as the 3D cell array primitive except that modeling coordinate positions are specified using only *x*- and *y*-coordinates, and the *z*-coordinate is always assumed to be zero. In addition, the cell array is defined by two points which define a rectangle that is taken to be aligned with the modeling coordinate axes.

If any color index is not defined, color index one is used. If color index one is not defined, the default values in Appendix D are used.

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

Extended cell array 3D

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

When processed by a renderer, this command has the same effect as the "cell array 3D" primitive, except that the colors may be passed as indexed or direct color values, depending on the setting of *color_type*.

If any color index is not defined, color index one is used. If color index one is not defined, the default values in Appendix D are used.

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

GDP 3D

gdp_id : INT32
points : LISTofCOORD_3D
data : LISTofCARD8

When processed by a renderer, the effect of this command is implementation-dependent. Because of floating point and color format discrepancies across a network interface, it is not anticipated that the GDP 3D will provide a useful extension mechanism, but it is provided for PHIGS compatibility purposes.

If the specified GDP identifier is not supported, the output command is ignored. Note that byte-swapping and floating point conversion may not be done on this type of output command.

GDP 2D

gdp_id : INT32
points : LISTofCOORD_2D
data : LISTofCARD8

When processed by a renderer, the effect of this command is implementation-dependent. Because of floating point and color format discrepancies across a network interface, it is not anticipated that the GDP 2D will provide a useful extension mechanism, but it is provided for PHIGS compatibility purposes.

If the specified GDP identifier is not supported, the output command is ignored. Note that byte-swapping and floating point conversion may not be done on this type of output command. |

Noop |

When processed by a renderer, this command will only update the current path. It is useful for client-side accumulate state traversals since it does not generate any output. |

4. Lookup Tables

A *lookup table* is a PEX resource that allows clients to create tables of values for various purposes. Lookup tables are used to support a level of indirection for some output primitive attributes as well as for storing view information, light source descriptions, depth-cue information, etc.

Tables may be sparse, therefore tables consist of index/entry pairs. The *index* is the number that will be used to reference that entry. Since indices are 16-bit values, index values are allowed to be any value in the range [0..65535], with the possible exceptions of 0 (some tables allow an index of zero, others do not) and 65535 (tables can have at most $2^{16} - 1$ definable entries, so if a table begins at zero, the maximum index value is 65534). An *entry* is the collection of information (or the data record) that is defined for each type of table. The table descriptions in this section include the definition of an entry for each type of table. A table index refers to an *undefined entry* if no table entry has ever been associated with that index, or if the associated table entry has been deleted. A table entry may contain one or more data items, depending on the table type. For instance, in the *MarkerBundle* table type, each entry in the table consists of a marker type value, a marker scale value, and a marker color value.

A lookup table may have *predefined entries*. Predefined entries are a set of contiguous table entries that are automatically filled in by the server when the table is created. **PEXGetTableInfo** returns the range of predefined entries. Predefined table entries may be deleted or overwritten. |

Each type of table has a specific index value that indicates the *default table entry*. If a table index references an undefined table entry, the contents of the default table entry will be used. If the default table entry is undefined, then the default attribute values (as listed in Appendix D) will be used as the default entry. The default entry for all tables is one, except for the view, depth cue, and color approximation tables whose default entry is zero.

PEX lookup tables are designed to support PHIGS set/realized semantics. A lookup table entry may be set to a value that is impossible to represent exactly. Such a value will be silently mapped to a reasonable default when rendering to the drawable. When retrieving entry values from a lookup table, clients may request the value as originally specified by the client (*value_type = Set*), or the value that is actually used when rendering, whether it is the value originally specified or an approximation (*value_type = Realized*). Since all predefined entries are by definition realizable, predefined lookup table entries will return the same value regardless of whether the *Set* or the *Realized* value is requested.

The allowable table types, the range of allowable index values, the default entry, and the format of a table entry for each table type are as follows:

LineBundle (1..65535, default entry = 1)

This type of lookup table is used to maintain attributes for drawing polyline and curve primitives. Depending on the setting of the aspect source flag attributes, polyline and curve attributes may be obtained from a line bundle table. Each entry in this type of table consists of the following:

line_type	LINE_TYPE
polyline_interp	POLYLINE_INTERP
curve_approx	CURVE_APPROX
line_width	FLOAT
line_color	COLOR_SPECIFIER

The attributes stored in a line bundle table are defined and used in the same fashion as the pipeline context attributes of the same name.

MarkerBundle (1..65535, default entry = 1)

This type of lookup table is used to maintain attributes for drawing marker primitives. Depending on the setting of the aspect source flag attributes, marker attributes may be obtained from a marker bundle table.

Each entry in this type of table consists of the following:

marker_type	MARKER_TYPE
marker_scale	FLOAT
marker_color	COLOR_SPECIFIER

The attributes stored in a marker bundle table are defined and used in the same fashion as the pipeline context attributes of the same name.

TextBundle (1..65535, default entry = 1)

This type of lookup table is used to maintain attributes for drawing text and annotation text primitives. Depending on the setting of the aspect source flag attributes, text and annotation text attributes may be obtained from a text bundle table. Each entry in this type of table consists of the following:

text_font_index	TABLE_INDEX
text_precision	TEXT_PRECISION
char_expansion	FLOAT
char_spacing	FLOAT
text_color	COLOR_SPECIFIER

The attributes stored in a text bundle table are defined and used in the same fashion as the pipeline context attributes of the same name.

InteriorBundle (1..65535, default entry = 1)

This type of lookup table is used to maintain attributes for drawing surface primitives. Depending on the setting of the aspect source flag attributes, surface attributes may be obtained from an interior bundle table. Each entry in this type of table consists of the following:

interior_style	INTERIOR_STYLE
interior_style_index	TYPE_OR_TABLE_INDEX
surface_color	COLOR_SPECIFIER
reflection_attr	REFLECTION_ATTR
reflection_model	REFLECTION_MODEL
surface_interp	SURFACE_INTERP
bf_interior_style	INTERIOR_STYLE
bf_interior_style_index	TYPE_OR_TABLE_INDEX
bf_surface_color	COLOR_SPECIFIER
bf_reflection_attr	REFLECTION_ATTR
bf_reflection_model	REFLECTION_MODEL
bf_surface_interp	SURFACE_INTERP
surface_approx	SURFACE_APPROX

The attributes stored in an interior bundle table are defined and used in the same fashion as the pipeline context attributes of the same name.

EdgeBundle (1..65535, default entry = 1)

This type of lookup table is used to maintain attributes for drawing edges of surface primitives. Depending on the setting of the aspect source flag attributes, surface edge attributes may be obtained from an edge bundle table. Each entry in this type of table consists of the following:

surface_edges	SWITCH
surface_edge_type	SURFACE_EDGE_TYPE
surface_edge_width	FLOAT
surface_edge_color	COLOR_SPECIFIER

The attributes stored in an edge bundle table are defined and used in the same fashion as the pipeline context attributes of the same name.

Pattern (1..65535, default entry = 1)

This type of lookup table can be used to maintain patterns for use when *interior_style* is set to *Pattern*.

color_type	COLOR_TYPE
numx	CARD16
numy	CARD16
colors	LISTofCOLOR

A pattern rectangle is comprised of *numx* × *numy* cells. *Color_type* indicates whether the color values are stored as indexed or direct color values. The colors are stored in the array row-by-row. The upper left hand cell in the pattern rectangle is the first one in the list of colors, followed by the remaining cells in the first row. The color values for cells in the second row follow, and so on.

Color (0..65534, default entry = 1)

This type of lookup table can be used to resolve indirect color references. Consequently, all color values in this type of table must be specified as direct colors.

color_type	COLOR_TYPE
color	DIRECT_COLOR

A *ColorType* error is generated if an attempt is made to set an entry with a *color_type* of *Indexed*.

TextFont (1..65535, default entry = 1)

This type of lookup table is used to maintain a list of font groups. Each font group is a list of resource IDs for either PEX fonts or X11 fonts. Resource IDs for fonts can appear multiple times in a font table, or even within a single table entry. Only PEX fonts are guaranteed to fully realize all of the PEX text attributes. Specifically, scaling and rotation operations on text strings are not guaranteed to affect text primitives if an X11 font is used, but they are guaranteed to work if a PEX font is used. Font values are specified as indices when using output commands. A font index can be used with a table of this type in order to obtain the actual font group to be used. Switching between fonts in the font group is accomplished by means of a switching mechanism embedded within strings that are to be rendered. It is up to the client to ensure that all of the character sets that are available in a table of this type are available in the default table entry as well.

font	LISTofFONT_ID

If a font is opened, bound to a font table, and then closed, the contents of the font will remain, since the contents are still being referenced by a font table. However, when the font table is queried, the value *AlreadyFreed* will be returned for such a font, since it no longer has a valid resource ID by which it can be referenced.

View (0..65534, default entry = 0)

This type of lookup table is used to maintain viewing information. "Views" are then specified as indices, which are used to look up the appropriate information in a view lookup table. *

clip_flags	BITMASK_SHORT
clip_limits	NPC_SUBVOLUME
orientation	MATRIX
mapping	MATRIX

Orientation is a matrix which maps geometry from the world coordinate system to the viewing reference coordinate system. (This is a right-handed coordinate system and is also known as the eye or viewing coordinate system.) *Mapping* is a matrix which maps geometry from the viewing reference coordinate system to the normalized projection coordinate system (also a right-handed coordinate system). This transformation is typically a parallel or perspective projection. The *clip_limits* specify the minimum and maximum of a rectangular volume in normalized projection coordinates and, *clip_flags* contains three bits that indicate whether or not clipping should be performed against the sides, back, and front planes of the volume specified by *clip_limits*. The actual clipping volume for each view is a rectangular parallelpiped that is computed by taking the intersection of the view clipping volume (determined by *clip_limits* and *clip_flags*) and the NPC subvolume. |

Light (1..65535, default entry = 1)

This type of lookup table is used to maintain light source definitions for use in light source shading computations. *

light_type	LIGHT_TYPE
direction	VECTOR_3D
point	COORD_3D
concentration	FLOAT
spread_angle	FLOAT
attenuation	[factor1, factor2 : FLOAT]
color	COLOR_SPECIFIER

There are four types of light sources currently supported in PEX: *ambient*, *WCS vector*, *WCS point* and *WCS spot*. Depending on the type of light, some of the values in a table entry may be ignored. |

An ambient light source is defined by *color*. It is a non-directional light that affects the color of all surfaces regardless of their orientation. Each ambient light causes a constant amount of intensity to be added to each point in the scene. |

A WCS vector light source (or directional light source) is defined by *direction* and *color*. A light source of this type is located at infinity so all light rays which emanate from it are parallel when they strike an objects surface. |

A WCS point light source (or a positional light source) illuminates equally in all directions and is specified by *point*, *attenuation*, and *color*. *point* indicates the world coordinate source of the light and *attenuation* indicates how the intensity of the point light source drops off as a function of distance. |

A WCS spot light source simulates the behavior of a spotlight and is specified by *direction*, *point*, *concentration*, *spread_angle*, *attenuation*, and *color*. *point* indicates the world coordinate source of the light and *direction* and *spread_angle* specify a cone of influence. The direction vector is relative to the specified light source position and *spread_angle* is the half-angle of the cone. The contribution of a spotlight is zero if the point in question lies outside of the cone of influence. |

DepthCue (0..65534, default entry = 0)

This type of lookup table is used to maintain depth-cueing information. *

mode	SWITCH
front_plane	FLOAT
back_plane	FLOAT
front_scaling	FLOAT
back_scaling	FLOAT
color	COLOR_SPECIFIER

The depth cueing computation modifies colors according to the distance from the viewing position. *color* indicates the color to which objects will tend when going from the *front_plane* to the *back_plane*. *front_scaling* and *back_scaling* represent values by which the colors will be scaled at the front and back depth cueing planes. The depth cueing computation is performed in NPC space-- *front_plane* is the distance to the |

front depth-cueing plane in NPC coordinates and *back_plane* is the distance to the back depth-cueing plane in NPC coordinates.

ColorApprox (0..65534, default entry = 0)

This type of lookup table is used to define the way that a renderer will transform rendering pipeline color values into displayable pixel values. Each entry in this type of table contains the following data:

type	COLOR_APPROX_TYPE
color_model	COLOR_APPROX_MODEL
max1	CARD16
max2	CARD16
max3	CARD16
mult1	CARD32
mult2	CARD32
mult3	CARD32
weight1	FLOAT
weight2	FLOAT
weight3	FLOAT
base_pixel	CARD32
dither	SWITCH

After a renderer has performed illumination, depth-cueing, and clipping operations, it is left with a rendering pipeline color that must be converted to a displayable pixel value. The renderer's current *color_approx_index* is used to determine which entry in a table of this type is to be used to perform the conversion to a displayable pixel value. As the color value emerges from the rendering pipeline, it is first converted to a color in the color space specified by *color_model*. (In the case where the rendering pipeline colors are already in the specified color space, this is a null mapping.)

If *type* is *ColorSpace*, each component of the converted color (*c1, c2, c3*) is scaled by the corresponding maximum value (*max1, max2, max3*). (*max1, max2, max3* can be used by the client to indicate the number of entries in the colormap for each color axis, minus 1. For example, to compute a pixel value for a 3-3-2 RGB colormap allocation, the max values would be 7, 7, and 3). The resulting color components are packed into a single pixel value by multiplying the first component by *mult1*, the second by *mult2*, the third by *mult3*, and then adding this to the *base_pixel*. When the *type* is *ColorSpace*, *weight1*, *weight2*, and *weight3* values are not used.

For a more concrete example, assume that the *color_model* is set to *RGB*. During the color approximation stage, each color value that emerges from the rendering pipeline must first be converted to an RGB triple. If the rendering pipeline performs color computations in RGB space then this conversion is a no-op. Each of the three components are then mapped to an integer value as follows:

red = red intensity component mapped into the range [0, *max1*]
green = green intensity component mapped into the range [0, *max2*]
blue = blue intensity component mapped into the range [0, *max3*]

and a single pixel value is formed by computing:

$$\text{pixel} = \mathit{mult1} \times \text{red} + \mathit{mult2} \times \text{green} + \mathit{mult3} \times \text{blue} + \mathit{base\_pixel}$$

If *color_type* is *ColorRange*, the values are combined in a different fashion during the color approximation stage. Once again, the rendering pipeline color is first converted into a color in the color space specified by *color_model*. First, the color components (*c1, c2, c3*) that emerge from the rendering pipeline are multiplied by their corresponding normalized weight values (*weight1, weight2, weight3*) and the terms are added together to form a single value. The weight values can be adjusted to allow equal weighting of the components (weights are all equal) or to minimize or eliminate one or more of the components (one or more weights equal to 0). For instance, the weight values of 0.30, 0.59, 0.11 can be used to convert an RGB value to a single-valued intensity according to the NTSC color standard.

Next, the computed value is multiplied by *max1* (*max2* and *max3* are not used) which should be set by the client to represent the number of color map entries in the range, minus 1. For instance, if a client desires to display its computed image on a pseudo-color display using gray scale, it could allocate 100 contiguous color cells in the color map, and set *max1* to the value of 99, so that intensity values would be mapped into the range [0,99]. This value is then replicated for each of the three color components, which are then multiplied by *mult1*, *mult2*, and *mult3* respectively. The values are then added together with *base_pixel* to form the pixel value that is to be written. It is up to the client to provide reasonable multipliers for drawables that are not three-channel in nature. For instance, multipliers of (1.0, 0.0, 0.0) could be used if the target drawable was known to be a window of visual type *PseudoColor*. PEX implementations are free to optimize the case where one or more of the multipliers is zero.

The use of addition rather than logical OR for composing pixel values permits allocations where the primary components are not allocated into distinct bitplanes. Since some hardware allows a performance improvement if the multiplication values (*mult1*, *mult2*, *mult3*) are powers of two, the **PEXGetImpDepConstants** request can be used to obtain the constant *BestColorApproxValues* in order to get an indication as to whether powers of two are preferred.

Dither is treated as a hint to the renderer as to whether or not some attempt at dithering should be performed. Whether or not dithering is supported and the dithering algorithm that is used are implementation-dependent (see **PEXGetImpDepConstants**).

4.1. Lookup Table Resource Management

The lookup table is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation, freeing, and copying of lookup table resources.

4.1.1. Create Lookup Table

Name:

PEXCreateLookupTable

Request:

drawable_example : DRAWABLE_ID
lut_id : LOOKUP_TABLE_ID
table_type : TABLE_TYPE

Errors:

IDChoice, Drawable, Match, Value, Alloc, LookupTable |

Description:

This request creates a lookup table resource for the specified *lut_id*, for use with drawables with the same root window and depth as the example drawable specified by *drawable_example*. The *table_type* parameter indicates the type of lookup table that is to be created. Some entries of a lookup table may be defined at the time the resource is created. The number of predefined entries and their contents are dependent on the type of table and are implementation-dependent.

4.1.2. Copy Lookup Table

Name:

PEXCopyLookupTable

Request:

src_lut_id : LOOKUP_TABLE_ID
dest_lut_id : LOOKUP_TABLE_ID

Errors:

LookupTable, Match

Description:

This request copies the source lookup table *src_lut_id* to a destination lookup table *dest_lut_id*, after first deleting all the entries in the destination lookup table. The *dest_lut_id* must already exist as a valid resource and it must be of the same type as *src_lut_id*.

4.1.3. Free Lookup Table

Name:

PEXFreeLookupTable

Request:

lut_id : LOOKUP_TABLE_ID

Errors:

LookupTable

Description:

This request deletes the association between the resource ID and the lookup table. The lookup table storage will be freed when no other resource references it.

4.2. Lookup Table Inquiry

These requests inquire lookup table attributes.

4.2.1. 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, Match, Value, LookupTable |

Description:

This request will return information about lookup tables of the specified *table_type* if they were to be used with drawables of the same root and depth as *drawable_example*. *Definable_entries* is the maximum number of entries that can be defined in this type of table, and includes the number of predefined entries. Predefined entries can be redefined by the client. All the entries between *predefined_min* and *predefined_max* are guaranteed to be defined initially (i.e., predefined entries must have contiguous indices). If *num_predefined* is zero, then the values for *predefined_min* and *predefined_max* are meaningless.

4.2.2. Get Predefined Entries

Name:

PEXGetPredefinedEntries

Request:

fp_format : FLOAT_FORMAT
drawable_example : DRAWABLE_ID
table_type : TABLE_TYPE
start : TABLE_INDEX
count : CARD16

Reply:

entries : LISTofTABLE_ENTRY

Errors:

Drawable, Match, Value, FloatingPointFormat, LookupTable |

Description:

This request will return the values for predefined table entries of the specified *table_type* if they were to be used with drawables of the same root and depth as *drawable_example*. *Count* table entries will be returned * in *entries*, starting with the table entry specified by *start*. The values in *entries* will be in the format defined for *table_type*. Floating point values in *entries* will be returned in the floating point format specified in *fp_format*.

4.2.3. Get Defined Indices

Name:

PEXGetDefinedIndices

Request:

lut_id : LOOKUP_TABLE_ID

Reply:

defined_indices : LISTofTABLE_INDEX

Errors:

LookupTable

Description:

This request will return in *defined_indices* a list of all the indices that are currently defined in the lookup table resource specified by *lut_id*. The entries returned include those predefined by the server and those that have been defined by clients.

4.2.4. Get Table Entry

Name:

PEXGetTableEntry

Request:

fp_format : FLOAT_FORMAT
lut_id : LOOKUP_TABLE_ID
index : TABLE_INDEX
value_type : {*Set, Realized*}

Reply:

status : {*Defined, Default*}
table_type : TABLE_TYPE
entry : TABLE_ENTRY

Errors:

LookupTable, FloatingPointFormat, Value

Description:

This request will return the type of lookup table and the lookup table entry specified by *index*. The entry will be obtained from the lookup table specified by *lut_id* and will be of the format indicated by *lut_id*'s table type. If the specified entry in the lookup table is not defined, the values for the default entry for that table type will be returned in *entry* and *status* will be set to *Default*. If the specified entry is defined, its contents will be returned in *entry* and *status* will be set to *Defined*. If *value_type* is *Set*, the values returned will be those originally specified by the client. If *value_type* is *Realized*, the values returned will be the values that are actually used during rendering (i.e., the default values, which are used if the specified value is not supported on the drawable). Floating point values in *entry* will be returned in the floating point format specified in *fp_format*.

4.2.5. Get Table Entries

Name:

PEXGetTableEntries

Request:

fp_format : FLOAT_FORMAT
lut_id : LOOKUP_TABLE_ID
start : TABLE_INDEX
count : CARD16
value_type : {*Set, Realized*}

Reply:

table_type : TABLE_TYPE
entries : LISTofTABLE_ENTRY

Errors:

LookupTable, Value, FloatingPointFormat

Description:

This request will return the type of table and *count* table entries from the lookup table specified by *lut_id*, starting at the entry specified by *start*. The default entry will be returned for any entry in the requested range that is not defined. If *value_type* is *Set*, the values returned will be those originally specified by the client. If *value_type* is *Realized*, the values returned will be the values that are actually used during rendering (i.e., the default values, which are used if the specified value is not supported on the drawable). Floating point values in *entries* will be returned in the floating point format specified in *fp_format*.

4.3. Lookup Table Modification

This section contains requests that modify lookup table resources.

4.3.1. Set Table Entries

Name:

PEXSetTableEntries

Request:

fp_format : FLOAT_FORMAT
lut_id : LOOKUP_TABLE_ID
start : TABLE_INDEX
count : CARD16
entries : LISTofTABLE_ENTRY

Errors:

LookupTable, Value, FloatingPointFormat, ColorType, Alloc

Description:

This request will set *count* lookup table entries in the lookup table resource specified by *lut_id*, starting at the entry indicated by *start*. The values to use when setting the entries in the lookup table are provided in *entries*, and are in the format specified for a lookup table of *lut_id*'s type. Floating point values in *entries* will be in the floating point format specified in *fp_format*.

4.3.2. Delete Table Entries

Name:

PEXDeleteTableEntries

Request:

lut_id : LOOKUP_TABLE_ID
start : TABLE_INDEX
count : CARD16

Errors:

LookupTable, Value

Description:

This request will delete the defined table entries in the lookup table resource specified by *lut_id*, between and including the entry specified by *start* and the entry specified by *start*+*count*-1.

5. Pipeline Contexts

A *pipeline context* is a PEX resource that contains an instance of the attributes that describe a rendering pipeline. The attributes in a pipeline context are copied to a renderer resource whenever a **PEXBeginRendering** request is executed. This section describes pipeline context attributes and the operations that can be performed on pipeline context resources.

Some of the requests in this section affect attributes of a pipeline context. The *item_mask* and *item_list* parameters specify which components are to be affected. Each bit in the *item_mask* indicates whether or not the corresponding attribute is affected. In the cases where pipeline context attributes are being set or queried, there is a corresponding entry in the *item_list* for each set bit in *item_mask*. It is therefore possible to affect one or many pipeline context attributes with a single request.

A name set resource ID is one of the attributes of a pipeline context. If a name set is created, bound to a pipeline context, and then freed, the contents of the name set will remain, since the contents are still being referenced by the pipeline context. However, when a pipeline context's attributes are 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 pipeline context components are listed in the following table.

Attribute Name	Data Type	Default Value
Marker attributes		
marker_type	MARKER_TYPE	*MarkerAsterisk*
marker_scale	FLOAT	1.0
marker_color	COLOR_SPECIFIER	{*Indexed*, 1}
marker_bundle_index	TABLE_INDEX	1
Text attributes		
text_font_index	TABLE_INDEX	1
text_precision	TEXT_PRECISION	*String*
char_expansion	FLOAT	1.0
char_spacing	FLOAT	0.0
text_color	COLOR_SPECIFIER	{*Indexed*, 1}
char_height	FLOAT	0.01
char_up_vector	VECTOR_2D	<0.0, 1.0>
text_path	TEXT_PATH	*PathRight*
text_alignment	TEXT_ALIGNMENT	{*HalignNormal, ValignNormal*}
atext_height	FLOAT	0.01
atext_up_vector	VECTOR_2D	<0.0, 1.0>
atext_path	TEXT_PATH	*PathRight*
atext_alignment	TEXT_ALIGNMENT	{*HalignNormal, ValignNormal*}
atext_style	ATEXT_STYLE	*ATextNotConnected*
text_bundle_index	TABLE_INDEX	1
Line and curve attributes		
line_type	LINE_TYPE	*LineTypeSolid*
line_width	FLOAT	1.0
line_color	COLOR_SPECIFIER	{*Indexed*, 1}
curve_approx	CURVE_APPROX	{1, 1.0}
polyline_interp	POLYLINE_INTERP	*PolylineInterpNone*
line_bundle_index	TABLE_INDEX	1
Surface attributes		
interior_style	INTERIOR_STYLE	*InteriorStyleHollow*
interior_style_index	TYPE_OR_TABLE_INDEX	1

surface_color	COLOR_SPECIFIER	{*Indexed*, 1}
reflection_attr	REFLECTION_ATTR	{1.0, 1.0, 1.0, 0.0, 0.0, {*Indexed*, 1}}
reflection_model	REFLECTION_MODEL	*ReflectionNoShading*
surface_interp	SURFACE_INTERP	*SurfaceInterpNone*
bf_interior_style	INTERIOR_STYLE	*InteriorStyleHollow*
bf_interior_style_index	TYPE_OR_TABLE_INDEX	1
bf_surface_color	COLOR_SPECIFIER	{*Indexed*, 1}
bf_reflection_attr	REFLECTION_ATTR	{1.0, 1.0, 1.0, 0.0, 0.0, {*Indexed*, 1}}
bf_reflection_model	REFLECTION_MODEL	*ReflectionNoShading*
bf_surface_interp	SURFACE_INTERP	*SurfaceInterpNone*
surface_approx	SURFACE_APPROX	{1, 1.0, 1.0}
culling_mode	CULL_MODE	*None*
distinguish	BOOLEAN	*False*
pattern_size	VECTOR_2D	{1.0, 1.0}
pattern_ref_pt	COORD_3D	{0.0, 0.0, 0.0}
pattern_ref_vec1	VECTOR_3D	<1.0, 0.0, 0.0>
pattern_ref_vec2	VECTOR_3D	<0.0, 1.0, 0.0>
interior_bundle_index	TABLE_INDEX	1
Surface edge attributes		
surface_edges	SWITCH	*Off*
surface_edge_type	SURFACE_EDGE_TYPE	*SurfaceEdgeSolid*
surface_edge_width	FLOAT	1.0
surface_edge_color	COLOR_SPECIFIER	{*Indexed*, 1}
edge_bundle_index	TABLE_INDEX	1
Geometry transformation attributes		
local_transform	MATRIX	identity
global_transform	MATRIX	identity
model_clip	CLIP_INDICATOR	*NoClip*
model_clip_volume	LISTofHALF_SPACE	*Null*
view_index	TABLE_INDEX	0
Color transformation attributes		
light_state	LISTofTABLE_INDEX	*Null*
depth_cue_index	TABLE_INDEX	0
color_approx_index	TABLE_INDEX	0
rdr_color_model	COLOR_MODEL	0
psurf_char	PSURF_CHAR	{1, *NULL*}
ASF attributes		
asfs	ASF_SPECIFIER	{0, all *Individual*}
Miscellaneous attributes		
pick_id	CARD32	0
HLHSR_identifier	CARD32	0
name_set †	NAME_SET_ID	*None*

The bits in the *asfs* field of an ASF_SPECIFIER are defined as:

marker_type_asf	ASF_VALUE	*Individual*
marker_scale_asf	ASF_VALUE	*Individual*
marker_color_asf	ASF_VALUE	*Individual*
text_font_index_asf	ASF_VALUE	*Individual*
text_precision_asf	ASF_VALUE	*Individual*

† When pipeline context attributes are copied to a renderer (e.g., whenever a **PEXBeginRendering** request occurs), the actual *contents* of the name set resource is copied, and not the resource ID of the name set.

char_expansion_asf	ASF_VALUE	*Individual*
char_spacing_asf	ASF_VALUE	*Individual*
text_color_asf	ASF_VALUE	*Individual*
line_type_asf	ASF_VALUE	*Individual*
line_width_asf	ASF_VALUE	*Individual*
line_color_asf	ASF_VALUE	*Individual*
curve_approx_asf	ASF_VALUE	*Individual*
polyline_interp_asf	ASF_VALUE	*Individual*
interior_style_asf	ASF_VALUE	*Individual*
interior_style_index_asf	ASF_VALUE	*Individual*
surface_color_asf	ASF_VALUE	*Individual*
reflection_model_asf	ASF_VALUE	*Individual*
surface_interp_asf	ASF_VALUE	*Individual*
reflection_attr_asf	ASF_VALUE	*Individual*
bf_interior_style_asf	ASF_VALUE	*Individual*
bf_interior_style_index_asf	ASF_VALUE	*Individual*
bf_surface_color_asf	ASF_VALUE	*Individual*
bf_reflection_model_asf	ASF_VALUE	*Individual*
bf_surface_interp_asf	ASF_VALUE	*Individual*
bf_reflection_attr_asf	ASF_VALUE	*Individual*
surface_approx_asf	ASF_VALUE	*Individual*
surface_edges_asf	ASF_VALUE	*Individual*
surface_edge_type_asf	ASF_VALUE	*Individual*
surface_edge_width_asf	ASF_VALUE	*Individual*
surface_edge_color_asf	ASF_VALUE	*Individual*

The attributes of the pipeline context resource are defined as follows:

marker_type
This attribute contains the marker type to use when drawing marker primitives. See the "Extension Information" section for descriptions of the registered marker types.

marker_scale
This attribute contains the marker scale factor to use when drawing marker primitives.

marker_color
This attribute contains the color value to use when drawing marker primitives.

marker_bundle_index
This attribute contains the lookup table index to be used to obtain bundled marker attributes from the marker bundle table.

text_font_index
This attribute contains the lookup table index to be used to obtain the font ID for drawing text and annotation text primitives.

text_precision
This attribute contains the text precision to use when drawing text and annotation text primitives.

char_expansion
This attribute contains the character expansion to use when drawing text primitives. The character expansion factor is the deviation of the width to height ratio of the characters from the ratio indicated by the font designer.

char_spacing

This attribute contains the character spacing to use when drawing text primitives. Character spacing specifies how much additional space is to be inserted between two adjacent character bodies and is specified as a fraction of the font-nominal character height.

text_color

This attribute contains the color value to use when drawing text and annotation text primitives.

char_height

This attribute contains the character height to use when drawing text primitives. The character height is specified in modeling coordinates.

char_up_vector

This attribute contains the character up vector to use when drawing text primitives. The up vector is specified in the text local coordinate system. The axes of this coordinate system are determined by the direction vectors specified with the text primitive.

text_path

This attribute contains the text path to use when drawing text primitives (i.e., the writing path of the text string).

text_alignment

This attribute contains the horizontal and vertical alignment to use when drawing text primitives.

atext_height

This attribute contains the character height to use when drawing annotation text primitives. The character height is specified in normalized projection coordinates.

atext_up_vector

This attribute contains the character up vector to use when drawing annotation text primitives. The up vector is specified in the annotation text local coordinate system, which is parallel to the display surface.

atext_path

This attribute contains the text path to use when drawing annotation text primitives (i.e. the writing path of the annotation text string).

atext_alignment

This attribute contains the horizontal and vertical alignment to use when drawing annotation text primitives.

atext_style

This attribute contains the annotation text style to use when drawing annotation text primitives. See the "Extension Information" section for descriptions of the registered annotation text styles.

text_bundle_index

This attribute contains the lookup table index to be used to obtain bundled text attributes from the text bundle table.

line_type

This attribute contains the type to use when drawing polyline and curve primitives. See the "Extension Information" section for descriptions of the registered line types.

line_width

This attribute contains the width to use when drawing polyline or curve primitives. This is the scale factor applied to the width of the polyline or curve primitive when it is to be rendered. Line width is applied in 2D raster coordinates after the primitive has been transformed from 3D space to 2D raster space.

line_color

This attribute contains the color value to use when drawing polyline and curve primitives.

curve_approx
This attribute contains the curve approximation to use when drawing curve primitives. It sets the curve approximation method and the tolerance value for drawing curves. See the "Extension Information" section for descriptions of the registered curve approximation methods and how the curve tolerance is used with each type.

polyline_interp
This attribute contains the polyline interpolation method to use when drawing polyline primitives. See the "Extension Information" section for descriptions of the registered polyline interpolation methods.

line_bundle_index
This attribute contains the lookup table index to be used to obtain bundled polyline and curve attributes from the line bundle table.

interior_style
This attribute contains the interior style to use when drawing surface primitives. See the "Extension Information" section for descriptions of the registered interior styles.

interior_style_index
This attribute contains the index to use if the interior style is of type *Pattern* or *Hatch*. The interior style index contains the table index of the pattern table entry to be used when the interior style is *Pattern*, and the hatch table enumerated type index to be used when the interior style is *Hatch*.

surface_color
This attribute contains the color value to use when drawing surface primitives.

reflection_attr
This attribute contains the ambient coefficient; the diffuse coefficient; the specular coefficient, concentration, and color; and the transmission coefficient that are to be used in the light source shading computations when rendering surfaces (area-defining primitives). The specular color attribute provides an additional coefficient per primary for use in the specular reflection computation. This allows highlights to be computed that are some color other than that of the light source. The specular concentration defines the sharpness of the specular highlights or the "shininess" of a surface. This value is typically used as the exponent in the specular reflection term of lighting equations and it ranges from zero to the maximum floating point value. If | *specular_conc* is zero, specular highlights are very broad. If *specular_conc* is much greater than zero, the highlights are very small and sharp, as if the surface was very shiny. The transmission coefficient indicates the amount of light that passes through a surface. A transmission coefficient of zero indicates that the surface is opaque (lets no light through). A transmission coefficient of 1.0 indicates that the surface is totally invisible (lets all light through).

reflection_model
This attribute contains the reflection model to use when drawing surface primitives. See the "Extension Information" section for descriptions of the registered reflection models.

surface_interp
This attribute contains the surface interpolation method to use when drawing surface primitives. See the "Extension Information" section for descriptions of the registered surface interpolation methods.

bf_interior_style
This attribute contains the interior style to use when drawing backfacing surface primitives. See the "Extension Information" section for descriptions of the registered interior styles.

bf_interior_style_index
This attribute contains the index to use for backfacing surface primitives if the interior style is of type *Pattern* or *Hatch*. The interior style index contains the table index of the pattern table entry to be used when the interior style is *Pattern*, and the hatch table enumerated type index to be used when the interior style is *Hatch*.

bf_surface_color
This attribute contains the color value to use when rendering backfacing surface primitives.

bf_reflection_attr
This attribute contains the reflection values to be used when rendering backfacing surfaces.

bf_reflection_model
This attribute contains the reflection model to use when drawing backfacing surface primitives. See the "Extension Information" section for descriptions of the registered reflection models.

bf_surface_interp
This attribute contains the surface interpolation method to use when drawing backfacing surface primitives. See the "Extension Information" section for descriptions of the registered surface interpolation methods.

surface_approx
This attribute contains the surface approximation and the tolerance to use when drawing surface primitives. See the "Extension Information" section for descriptions of the registered surface approximation methods and of how the surface tolerance is used with each of them.

culling_mode
This attribute contains the culling mode that is used in processing backfacing surfaces. If the culling mode is *BackFaces*, all back-facing surfaces will be culled and only front-facing surfaces will be rendered. If the culling mode is *FrontFaces*, all front-facing surfaces will be culled and only back-facing surfaces will be rendered. If the culling mode is *None*, both front- and back-facing polygons will be rendered.

distinguish
This attribute contains the distinguish mode that is used in processing backfacing surfaces. This flag selects whether back-facing surfaces are rendered with the back-face surface attributes or the front-face surface attributes. If distinguish is *True*, then back-face attributes are used to render the surface. If *distinguish* is *False*, then front-face attributes are used to render the surface.

pattern_size
This attribute contains the pattern size to use when drawing surface primitives. It is specified in modeling coordinates. The value <*x*,0> will be used as the pattern width vector and the value <0,*y*> will be used as the pattern height vector. If the interior style is *Pattern*, the renderer attempts to use these values, plus the pattern reference point and the pattern reference vectors, to position, scale, and rotate the pattern on the surface. |

pattern_ref_pt
This attribute contains the pattern reference point to use when drawing surface primitives. It is specified in modeling coordinates. When the interior style is *Pattern*, the renderer attempts to use the pattern reference point, reference vectors, and the pattern size to position and scale the pattern on the surface. |

pattern_ref_vec1

This attribute contains the first of two pattern reference vectors to be used when interior style is *Pattern*. It is specified in modeling coordinates. When the interior style is *Pattern*, the renderer attempts to use the two pattern reference vectors, the pattern reference point, and the pattern size to position, scale, and rotate the pattern on the surface. |

pattern_ref_vec2

This attribute contains the second of two pattern reference vectors to be used when interior style is *Pattern*. It is specified in modeling coordinates. When the interior style is *Pattern*, the renderer attempts to use the two pattern reference vectors, the pattern reference point, and the pattern size to position, scale, and rotate the pattern on the surface. |

interior_bundle_index
This attribute contains the lookup table index to be used to obtain bundled interior attributes from the interior bundle table.

surface_edges
This attribute contains the surface edge flag attribute, which is used to enable or disable surface edge drawing. If *surface_edges* is *Off*, surface edge drawing is disabled. If *surface_edges* is *On*, surface edge drawing is enabled. Surface edges are drawn using the surface edge color, surface edge type, and surface edge width.

surface_edge_type
This attribute contains the edge type to use when drawing surface edges. See the "Extension Information" section for descriptions of the registered surface edge types.

surface_edge_width
This attribute contains the edge width to use when drawing surface edges. This is the scale factor applied to the width of a surface edge when a surface edge is to be rendered. It is applied in 2D raster coordinates after a surface edge primitive has been transformed from 3D space to 2D raster space.

surface_edge_color
This attribute contains the color value to use when drawing surface edges.

edge_bundle_index
This attribute contains the lookup table index to be used to obtain bundled surface edge attributes from the surface edge bundle table.

local_transform
This attribute contains the local modeling transformation matrix that is used when drawing output primitives.

global_transform
This attribute contains the global modeling transformation matrix that is used when drawing output primitives.

model_clip
This attribute contains the model clipping flag that indicates whether or not to perform modeling clipping when drawing output primitives.

model_clip_volume
This attribute contains the model clipping volume that is used whenever modeling clipping is enabled.

view_index
This attribute contains the lookup table index to be used to obtain viewing attributes from the view table.

light_state
This attribute contains a list of table indices that specify those light sources that are enabled ("turned on"). Any light whose table index is not in this list is considered disabled ("turned off").

depth_cue_index
This attribute contains the lookup table index to be used to obtain bundled depth-cue attributes from the depth-cue bundle table.

color_approx_index
This attribute contains the lookup table index to be used to obtain the color approximation parameters from the color approximation table.

rdr_color_model
This attribute contains the rendering color model that is to be used during the interpolation of shaded primitives. See the "Extension Information" section for descriptions of the registered rendering color models.

psurf_char
This attribute contains the parametric surface characteristics that are to be used when rendering surfaces. The *psc_type* field specifies the parametric surface characteristics type to be used when rendering surfaces. The *psc_data* field supplies additional data that may be used. See the "Extension Information" section for descriptions of the registered parametric surface characteristics types.

asfs

This attribute contains an aspect source flag (asf) for each attribute that can be obtained from a bundle lookup table. When rendering, if the value for an asf is set to *Individual*, the value for the corresponding attribute will be obtained directly from the current value within the renderer. If the value for the asf is set to *Bundled*, the value for the attribute will be obtained from the bundle lookup table. When setting asfs, a separate mask (*enables*) is used to indicate which asfs are actually to be modified. The value to which an asf is to be modified will then be taken from the corresponding bit in the *asfs* bitmask. The *enables* field of the ASF_SPECIFIER is meaningful only when creating or changing attributes of a pipeline context. During copying, all asf values are copied, and when queried, all asf values are returned and a value with all defined asfs set is returned for the *enables* bitmask. (This implies that it is not necessary to consider the *enables* field to be part of the state that is saved and restored while rendering.)

pick_id

This attribute contains the pick ID, which is used in conjunction with picking operations.

HLHSR_identifier

This attribute contains an HLHSR identifier. It is a noop for all registered HLHSR modes except *ZBufferId*. |
When the renderer's *HLHSR_mode* is set to *ZBufferId* then a zero value will disable z-buffering and a value |
of one will enable z-buffering. For non-registered HLHSR modes the meaning is implementation dependent. |
Conceptually, this attribute is bound to all output primitives as they enter the rendering pipeline.

name_set

This attribute contains a reference to a name set resource. When attributes of a pipeline context are copied to a renderer, the *contents* of this name set are copied to the renderer.

5.1. Pipeline Context Resource Management

The pipeline context is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation, freeing, and copying of pipeline contexts.

5.1.1. Create Pipeline Context

Name:
PEXCreatePipelineContext

Request:
fp_format : FLOAT_FORMAT
pc_id : PIPELINE_CONTEXT_ID
item_mask : PC_BITMASK
item_list : LISTofVALUE

Errors:
IDChoice, Value, FloatingPointFormat, ColorType, Alloc, NameSet |

Description:

This request creates a pipeline context resource for the specified *pc_id*. The *item_mask* defines those pipeline context attributes that are to be explicitly set at the time the resource is created. The *item_list* contains the corresponding list of values used to modify the newly-created pipeline context. Floating point values in *item_list* will be in the floating point format specified in *fp_format*. Similarly, color values in *item_list* will be the color type specified by *color_type*.

5.1.2. Copy Pipeline Context

Name:
PEXCopyPipelineContext

Request:
src_pc_id : PIPELINE_CONTEXT_ID
dest_pc_id : PIPELINE_CONTEXT_ID
item_mask : PC_BITMASK

Errors:
PipelineContext, Value

Description:

This request copies the source pipeline context *src_pc_id* to a destination pipeline context *dest_pc_id*. The *dest_pc_id* must already exist as a valid resource. The *item_mask* indicates which values in the pipeline context will be copied.

5.1.3. Free Pipeline Context

Name:

PEXFreePipelineContext

Request:

pc_id : PIPELINE_CONTEXT_ID

Errors:

PipelineContext

Description:

This request deletes the association between the resource ID and the pipeline context. The pipeline context storage will be freed when no other resource references it.

5.2. Pipeline Context Inquiry

This section defines the requests that can be used to inquire pipeline context attributes.

5.2.1. Get Pipeline Context

Name:

PEXGetPipelineContext

Request:

fp_format : FLOAT_FORMAT
pc_id : PIPELINE_CONTEXT_ID
item_mask : PC_BITMASK

Reply:

item_list : LISTofVALUE

Errors:

PipelineContext, FloatingPointFormat, Value

Description:

This request will return components of the pipeline context specified by *pc_id*. The *item_mask* specifies which components are to be inquired and returned. The specified attributes of the pipeline context will be returned in *item_list*. Floating point values in *item_list* will be returned in the floating point format specified in *fp_format*.

5.3. Pipeline Context Modification

This section defines the requests that can be used to modify attributes of pipeline context resources.

5.3.1. Change Pipeline Context

Name:

PEXChangePipelineContext

Request:

fp_format : FLOAT_FORMAT
pc_id : PIPELINE_CONTEXT_ID
item_mask : PC_BITMASK
item_list : LISTofVALUE

Errors:

PipelineContext, Value, FloatingPointFormat, ColorType, NameSet |

Description:

This request will modify components of the pipeline context specified by *pc_id*. The *item_mask* specifies which components are to be modified. The values for the attributes that are to be modified are contained in *item_list*. Floating point values in *item_list* will be in the floating point format specified in *fp_format*. Similarly, color values in *item_list* will be the color type specified by *color_type*.

6. Renderers

A *renderer* is a PEX resource that can be created for the purpose of doing 3D rendering. A renderer consists of resource IDs for various tables and name sets, the resource ID of a pipeline context from which the initial rendering pipeline attributes will be copied, and other attributes. A renderer also manages the HLHSR buffer needed for some hidden line/hidden surface algorithms. A renderer is made ready for rendering by the **PEXBeginRendering** request (a **PEXBeginRendering** is also performed implicitly as part of the **PEXRenderNetwork** request). A PEX implementation may choose to allow certain renderer attributes to be bound only at the time of a **PEXBeginRendering** request or at anytime during the lifetime of the renderer. To obtain information about how renderer attributes are bound for a particular PEX implementation, the **PEXGetRendererDynamics** request should be used.

Some renderer resource requests require an *item_mask* parameter. Each bit in the *item_mask* indicates whether or not the corresponding attribute is to be set/queried. There is a corresponding entry in the *item_list* for each set bit in *item_mask*. It is therefore possible to set/query one or many renderer attributes with a single request.

Any of a renderer's attributes that are resource IDs that are not explicitly specified will be set to the default value of *Null* when the renderer is created. During rendering operations, if a renderer makes an attempt to access an attribute from a lookup table whose resource ID is *Null*, the default attributes for the lookup table (as listed in Appendix D) will be used. If a renderer makes an attempt to read from a name set whose resource ID is *Null*, the result will be as if the name set was empty. If the renderer's pipeline context attribute contains the resource ID *Null*, the default values for pipeline context attributes will be copied into the renderer whenever a **PEXBeginRendering** request is performed.

Pipeline context, lookup table, and name set resources can be bound to a renderer and then freed. When this happens, the contents of these resources will remain, since they are still being referenced by the renderer. However, when that renderer'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.

If a window that is associated with a renderer is destroyed or resized while the renderer is in the *Rendering* state, an implicit **PEXEndRendering** is performed by the server with *flush* equal to *False* in order to return the renderer to the *Idle* state. All subsequent output and traversal commands (e.g., **PEXRenderOutputCommands**, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**) are ignored, up to and including the final **PEXEndRendering**. No errors are generated; however, at a subsequent **PEXBeginRendering**, an *Alloc* error may be generated if an attempt to reallocate the z-buffer fails. |

If a window that is associated with a renderer is moved, exposed, or occluded while the renderer is in the *Rendering* state, the server must continue to process output commands using the newly-modified window hierarchy until the next explicit or implicit **PEXEndRendering** occurs. |

Whenever an implicit or explicit **PEXBeginRendering** occurs and the *HLHSR_mode* indicates z-buffering should be performed, a z-buffer suitable for use with the specified drawable will be allocated, if necessary, and bound to the renderer. (A new z-buffer is only allocated if a z-buffer does not exist for the renderer or, if the z-buffer which is bound to the renderer is not suitable for the current drawable.) If the z-buffer could not be allocated, an *Alloc* error is generated and the rendering is aborted. If *clear_Z* is enabled or, if a new z-buffer was allocated, then the z-buffer will be cleared to infinity in the region specified by the renderer's current clip list. The value of infinity is implementation-dependent. |

During rendering, primitives will be clipped by the renderer's clip list. If z-buffering is enabled, primitives closer to the eye will be drawn over primitives that are further away. When primitives have the same z values, it is implementation-dependent which primitive will get drawn.

The z-buffer will not be deallocated when an explicit or implicit **PEXEndRendering** occurs. It remains bound to the renderer until the renderer is freed. When the next **PEXBeginRendering** occurs, an attempt will be made to |

reuse the z-buffer. If it cannot be reused (for instance, if the previous and current drawable are a different size), it will be deallocated and a z-buffer suitable for the new drawable will be allocated.

The renderer components are listed in the following table. The abbreviation "imp. dep." means that the default value is implementation-dependent. |

Attribute Name	Data Type	Default Value
pipeline_context	PIPELINE_CONTEXT_ID	*Null*
current_path	LISTofELEMENT_REF	*Null*
renderer_state	RENDERER_STATE	*Idle*
marker_bundle	LOOKUP_TABLE_ID	*Null*
text_bundle	LOOKUP_TABLE_ID	*Null*
line_bundle	LOOKUP_TABLE_ID	*Null*
interior_bundle	LOOKUP_TABLE_ID	*Null*
edge_bundle	LOOKUP_TABLE_ID	*Null*
view_table	LOOKUP_TABLE_ID	*Null*
color_table	LOOKUP_TABLE_ID	*Null*
depth_cue_table	LOOKUP_TABLE_ID	*Null*
light_table	LOOKUP_TABLE_ID	*Null*
color_approx_table	LOOKUP_TABLE_ID	*Null*
pattern_table	LOOKUP_TABLE_ID	*Null*
text_font_table	LOOKUP_TABLE_ID	*Null*
highlight_inclusion	NAME_SET_ID	*Null*
highlight_exclusion	NAME_SET_ID	*Null*
invisibility_inclusion	NAME_SET_ID	*Null*
invisibility_exclusion	NAME_SET_ID	*Null*
pick_inclusion	NAME_SET_ID	*Null*
pick_exclusion	NAME_SET_ID	*Null*
pick_start_path	LISTofELEMENT_REF	*Null*
background_color	COLOR_SPECIFIER	{Indexed, 0}
clear_I	BOOLEAN	*False*
clear_Z	BOOLEAN	*True*
echo_mode	ECHO_MODE	*NoEcho*
HLHSR_mode	HLHSR_MODE	1
NPC_subvolume	NPC_SUBVOLUME	{(0.0,0.0,0.0),(1.0,1.0,1.0)}
viewport	VIEWPORT	{imp. dep., imp. dep., True}
clip_list	LISTofDEVICE_RECT	*Null*

The attributes of the renderer resource are defined as follows:

pipeline_context
This attribute specifies the resource ID of the pipeline context from which initial rendering pipeline attribute values will be copied whenever an explicit or implicit **PEXBeginRendering** request is executed.

current_path
This attribute contains a list of element references for keeping track of paths for client-side traversal.

renderer_state
This attribute contains the current state of the renderer. The *renderer_state* is set to *Rendering* whenever an explicit or implicit **PEXBeginRendering** request is processed, to *Picking* whenever an explicit or implicit **PEXBeginPickOne** or **PEXBeginPickAll** request is processed, and to *Idle* whenever a explicit or implicit **PEXEndRendering**, **PEXEndPickOne** or **PEXEndPickAll** request is processed. |

marker_bundle
This attribute contains the resource ID of the marker bundle lookup table to be used when rendering.

text_bundle
This attribute contains the resource ID of the text bundle lookup table to be used when rendering.

line_bundle
This attribute contains the resource ID of the line bundle lookup table to be used when rendering.

interior_bundle
This attribute contains the resource ID of the interior bundle lookup table to be used when rendering.

edge_bundle
This attribute contains the resource ID of the edge bundle lookup table to be used when rendering.

view_table
This attribute contains the resource ID of the view lookup table to be used when rendering.

color_table
This attribute contains the resource ID of the color lookup table to be used to resolve references to indexed colors.

depth_cue_table
This attribute contains the resource ID of the depth-cue lookup table to be used when rendering.

light_table
This attribute contains the resource ID of the light lookup table to be used when rendering.

color_approx_table
This attribute contains the resource ID of the color approximation lookup table to be used during the color approximation stage of the rendering pipeline.

pattern_table
This attribute contains the resource ID of the lookup table to be used when referencing patterns.

text_font_table
This attribute contains the resource ID of the lookup table to be used when referencing text fonts.

highlight_inclusion
This attribute contains the resource ID of the name set to be used as the highlight inclusion set.

highlight_exclusion
This attribute contains the resource ID of the name set to be used as the highlight exclusion set.

invisibility_inclusion
This attribute contains the resource ID of the name set to be used as the invisibility inclusion set.

invisibility_exclusion
This attribute contains the resource ID of the name set to be used as the invisibility exclusion set.

pick_inclusion |
This attribute contains the resource ID of the name set to be used as the pick inclusion filter during picking. |

pick_exclusion |
This attribute contains the resource ID of the name set to be used as the pick exclusion filter during picking. |

pick_start_path |
This attribute contains a hierarchical path that indicates where to start the next server side "pick all" traversal. |
PEXPickAll starts hit testing at the next element after the one specified by this path. **PEXPickAll** and |
PEXEndPickAll both set this attribute to the path of the last element processed during the "pick all" traversal. |

background_color |
This attribute contains the color value to use when clearing the image buffer. The image buffer is cleared by |
PEXBeginRendering when *clear_I* is enabled. The *color_approx_index* in the initial pipeline context is used |

to map the color to a pixel value.

clear_I

This attribute indicates whether the image buffer should be cleared to the background color at **PEXBeginRendering**. Only pixels within the renderer's clip list are cleared.

clear_Z

This attribute indicates whether the z-buffer should be cleared to infinity (the value used for infinity is platform dependent) at the start of rendering. Only pixels within the renderer's clip list are cleared. This attribute is ignored if the *HLHSR_mode* does not require a z-buffer.

echo_mode

This attribute indicates whether primitives should be echoed, unechoed or drawn normally. The method used for echoing and unechoing primitives is implementation dependent and there is no guarantee that the method employed will preserve temporal priority. Echoing takes precedence over highlighting. It is recommended, but not required, that the echo and highlight methods are visually distinct.

HLHSR_mode

This attribute contains the hidden line/hidden surface method used when resolving visibility of overlapping primitives. The **PEXGetRendererDynamics** request can be used to determine whether changing the *HLHSR_mode* while the *renderer_state* attribute is set to *Rendering* has any effect.

NPC_subvolume

This attribute contains the normalized project coordinates for the sub-volume space that is to be mapped to the viewport.

viewport

This attribute is used to describe the area within a drawable in which 3D graphics primitives may appear. The viewport coordinates are specified in device coordinates, which have an x,y offset (in pixels) from the lower left corner of the drawable. It is permissible to specify a viewport with boundaries outside of the drawable. Geometry that is mapped to values outside the drawable will be clipped. Viewport z values must be in the range [0-1.0], where z=0 maps to the device-coordinate z value representing objects that are furthest from the viewing position, and z=1 maps to the device-coordinate value representing objects that are closest to the viewing position. Depending on the dynamics of the *viewport* attribute (see **PEXGetRendererDynamics**), the values that define a viewport may be bound at any time, or they may take effect only at the time of a **PEXBeginRendering**. Whenever the viewport values are bound, the viewport's *use_drawable* flag is examined first. If it is set to *True*, the viewport width and height are obtained from the drawable's current width and height and the viewport z values are obtained from the values specified in the viewport structure. If the *use_drawable* flag is *False*, the viewport width, height and z values are all set to the values in the viewport structure. The viewport is then set to the largest rectangle, anchored at the lower left corner, that achieves an isotropic mapping to the renderer's *NPC_subvolume*. The viewport will remain in the same position relative to the lower left hand corner of the drawable after a resize event has occurred.

clip_list

This attribute contains a list of rectangles in device coordinates that define the portions of the drawable in which rendering is enabled. If the list is *Null*, then all of the pixels on the drawable may be overwritten during rendering. If the list is not *Null*, the renderer may only render on those pixels that are within the rectangles in the list. The rectangles should be non-overlapping, or the graphics results will be undefined. Pixels that are outside of all of the rectangles in *clip_list* are effectively "write-protected". The rectangles in the clip list are defined in device coordinates (0,0 is at the lower left hand corner of the window) and will remain in the same position relative to the lower left hand corner of the drawable after a resize has occurred. (Note: If a z-buffering algorithm is used, only those pixels under the rectangles in the clip list will have their z values initialized when a **PEXBeginRendering** or a **PEXRenderNetwork** request is issued.)

6.1. Renderer Resource Management

The renderer is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation and freeing of renderer resources.

6.1.1. 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, Drawable, PipelineContext, NameSet, LookupTable, FloatingPointFormat, Value, Alloc, Match

Description:

This request creates a renderer resource for the specified *rdr_id*. It can be used to render onto drawables with the same root window and depth as the example drawable specified by *drawable_example*. The *item_mask* defines those renderer attributes that are to be explicitly set at the time the resource is created. The *item_list* contains the corresponding list of values used to modify the newly-created renderer. Floating point values in *item_list* will be in the floating point format specified in *fp_format*.

6.1.2. Free Renderer

Name:

PEXFreeRenderer

Request:

rdr_id : RENDERER_ID

Errors:

Renderer

Description:

This request deletes the specified renderer resource and frees the storage associated with it. If the renderer's *renderer_state* attribute is set to *Rendering* when this request is processed, an implicit **PEXEndRendering** request with *flush* equal to *False* will be performed before the renderer is freed. If the renderer's *renderer_state* attribute is set to *Picking* when this request is processed, the pick operation will be aborted. |

6.2. Renderer Modification

The requests in this section can be used to modify attributes of renderer resources.

6.2.1. Change Renderer

Name:

PEXChangeRenderer

Request:

fp_format : FLOAT_FORMAT
rdr_id : RENDERER_ID
item_mask : BITMASK
item_list : LISTofVALUE

Errors:

Renderer, Match, Value, FloatingPointFormat, NameSet, LookupTable, PipelineContext

Description:

This request changes components of a renderer. The *item_mask* and *item_list* specify which components are to be changed. Each bit in the *item_mask* indicates whether or not there is a corresponding entry in the *item_list*. It is therefore possible to modify one or many renderer attributes with a **PEXChangeRenderer** request. A renderer's *current_path* and *renderer_state* attribute are read-only, therefore attempts to modify them will be ignored. Floating point values in *item_list* will be in the floating point format specified in *fp_format*.

6.3. Renderer Inquiry

The requests in this section can be used to inquire renderer attributes.

6.3.1. Get Renderer Attributes

Name:

PEXGetRendererAttributes

Request:

fp_format : FLOAT_FORMAT
rdr_id : RENDERER_ID
item_mask : BITMASK

Reply:

item_list : LISTofVALUE

Errors:

Renderer, FloatingPointFormat, Value

Description:

This request will return components of the renderer specified by *rdr_id*. The *item_mask* specifies which components are to be inquired and returned. The specified attributes of the renderer will be returned in *item_list*. Floating point values in *item_list* will be returned in the floating point format specified in *fp_format*. (The values returned are those that were last set by the client. Depending on the dynamics of the renderer, these may or may not be the values currently in use. See **PEXGetRendererDynamics.)**

6.3.2. Get Renderer Dynamics

Name:

PEXGetRendererDynamics

Request:

rdr_id : RENDERER_ID

Reply:

tables : BITMASK
namesets : BITMASK
attributes : BITMASK

Errors:

Renderer

Description:

This request will return the dynamics (binding times) for all of the attributes of the specified renderer. The *tables* bitmask has the following bits defined:

MarkerBundle	*MarkerBundleContents*
TextBundle	*TextBundleContents*
LineBundle	*LineBundleContents*
InteriorBundle	*InteriorBundleContents*
EdgeBundle	*EdgeBundleContents*
ViewTable	*ViewTableContents*
ColorTable	*ColorTableContents*
DepthCueTable	*DepthCueTableContents*
LightTable	*LightTableContents*
ColorApproxTable	*ColorApproxTableContents*
PatternTable	*PatternTableContents*
TextFontTable	*TextFontTableContents*

The *namesets* bitmask has the following bits defined:

HighlightNameset	*HighlightNamesetContents*
InvisibilityNameset	*InvisibilityNamesetContents*
PickNameset	*PickNamesetContents*

The *attributes* bitmask has the following bits defined:

HLHSRMode
NPCSubvolume
Viewport
ClipList
EchoMode

For each defined bit, a value of zero indicates that the specified attribute is *dynamic* and a value of one indicates that it is not. Changes to *dynamic* attributes take effect immediately. Attributes which are not *dynamic* can be modified at any time but the new value will not take effect until the next explicit or implicit **PEXBeginRendering** occurs. Implementations that allow attributes such as *HLHSRMode* to be *dynamic* must specify the semantics of changing between all possible supported values for that attribute.

6.4. Client-Side Traversal Support

These requests provide support for client-side structure traversal. See section on **Renderer Picking** for requests to do client side picking. Client-side search traversals are not supported. |

6.4.1. Begin Rendering

Name:

PEXBeginRendering

Request:

rdr_id : RENDERER_ID
drawable_id : DRAWABLE_ID

Errors:

Renderer, Drawable, Match, Alloc, RendererState

Description:

This request causes rendering to begin on the drawable specified by *drawable_id*. The output of the renderer specified by *rdr_id* is bound to that drawable until a **PEXEndRendering** request is processed. The initial rendering pipeline attributes are copied into the renderer from the pipeline context specified by the renderer's *pipeline_context* attribute. If *pipeline_context* is *Null*, the default values for all of the attributes in a pipeline context are copied into the renderer instead. The renderer's *renderer_state* attribute is set to *Rendering* and its *current_path* attribute is initialized to structure id zero with an element offset of zero. If *clear_I* is enabled then the image buffer will be cleared to *background_color*. If *clear_Z* is enabled and the *HLHSR_mode* indicates that z-buffering should be performed, then the z-buffer will be cleared to infinity. The image and z-buffer are cleared only in the region specified by *clip_list*. | | |

If the specified drawable does not have the same root and depth as the drawable that was passed in to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRendererTargets**, a *Match* error will be generated. | | |

If *renderer_state* is currently *Rendering* or *Picking*, the operation is aborted, the **PEXBeginRendering** request is executed, and a *RendererState* error is returned. | |

6.4.2. End Rendering

Name:

PEXEndRendering

Request:

rdr_id : RENDERER_ID
flush : BOOLEAN

Errors:

Renderer, RendererState |

Description:

This request terminates the current rendering operation for the renderer specified by *rdr_id*. If *flush* is *True*, all pending output will be rendered onto the drawable. If *flush* is *False*, all pending output is discarded. In either case, the *renderer_state* attribute of the renderer is set to *Idle*. If the *renderer_state* | |

attribute is currently *Idle* (i.e., no rendering is in progress or the rendering was aborted due to a resize), the request is ignored and no error is generated. If the *renderer_state* is currently *Picking* then a *RendererState* error will be generated. |

6.4.3. Begin Structure *

Name:

PEXBeginStructure

Request:

rdr_id : RENDERER_ID
s_id : CARD32

Errors:

Renderer

Description:

If the *renderer_state* attribute of the renderer specified by *rdr_id* is set to *Rendering* or *Picking*, then this request causes the pipeline attributes of the renderer to be saved. If the *renderer_state* attribute is set to *Idle*, then this request is ignored. The attributes of the renderer resource itself (i.e., the attributes of a renderer that can be set/queried, including table and name set resource IDs) are not saved. The element offset of the last entry in the renderer's *current_path* is incremented (to account for the client-side "execute structure" command), then the name specified by *s_id* together with an element offset of zero (indicating an empty structure) is appended to the *current_path* attribute. Each subsequent output command will cause the element offset to be incremented by one until the next **PEXBeginStructure** request or the next **PEXEndStructure** request. |

After saving the current rendering pipeline attributes, the *global_transform* attribute is set to the matrix computed by concatenating the current *local_transform* and the current *global_transform* matrices. Then the *local_transform* matrix is set to the identity matrix.

6.4.4. End Structure

Name:

PEXEndStructure

Request:

rdr_id : RENDERER_ID

Errors:

Renderer, RendererState

Description:

If the *renderer_state* attribute of the renderer specified by *rdr_id* is set to *Rendering* or *Picking*, then this request restores the last-saved pipeline attributes in the renderer. If the *renderer_state* attribute is set to *Idle*, then this request is ignored. In addition, the last element reference in the renderer's *current_path* is removed, and subsequent output commands will cause the element offset of the element reference at the end of the list to be incremented. This request itself does not cause an increment to *current_path*. |

6.5. Rendering Commands

These requests cause output commands to be processed by a renderer.

6.5.1. Render Output Commands

Name:

PEXRenderOutputCommands

Request:

fp_format : FLOAT_FORMAT
rdr_id : RENDERER_ID
cmds : LISTofOUTPUT_CMD

Errors:

Renderer, FloatingPointFormat, OutputCommand

Description:

If the *renderer_state* attribute of the renderer specified by *rdr_id* is set to *Rendering* or *Picking*, the output commands in *cmds* will be immediately processed. If the *renderer_state* attribute is set to *Idle*, the commands will be ignored. Floating point values in *cmds* will be in the floating point format specified in *fp_format*. The formats for the various types of output commands can be found Section 3 - *Output Commands*. Output commands are processed by the server in order until one is found to be in error, or until the entire list has been processed. The erroneous output command and all others following it in the request are ignored, and an *OutputCommand* error is returned to the client. The *current_path* attribute of the renderer is updated to reflect only those output commands actually processed. |

6.5.2. Render Elements |

Name: |

PEXRenderElements |

Request: |

rdr_id : RENDERER_ID |
s_id : STRUCTURE_ID |
range : ELEMENT_RANGE |

Errors: |

Renderer, Structure, Value |

Description: |

If the *renderer_state* attribute of the renderer specified by *rdr_id* is set to *Rendering* or *Picking*, then this request causes a range of structure elements in the structure specified by *s_id* to be processed using the specified renderer. If the *renderer_state* attribute is set to *Idle*, then this request is ignored. The semantics of this request are the same as for **PEXRenderOutputCommands** except that the output commands are taken from the specified structure. |

The offsets that define the range of elements to be rendered are computed using the positions in *range* in the following fashion. If *whence* is *Beginning*, the computed offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the structure. *Offset* can be a negative number. |

If either computed offset is less than zero, it will be set to zero before the rendering occurs. If either computed offset is greater than the number of elements in the structure, it will be set to the offset of the last structure element. The structure's element pointer is not affected by this request. |

6.5.3. Accumulate State |

Name: |
PEXAccumulateState |

Request: |
rdr_id : RENDERER_ID |
path : LISTofELEMENT_REF |

Errors: |
Renderer, Path |

Description: |

If the *renderer_state* attribute of the renderer specified by *rdr_id* is set to *Rendering* or *Picking*, this request accumulates the state that would be in effect if a traversal were done to the element specified by *path*. If the *renderer_state* attribute is set to *Idle*, then this request is ignored. |

Accumulation of state begins with the current pipeline attributes. A linear descent down the specified path is then made and the structure elements that lie along the specified path are examined in order. Any element that contains an output command that would modify pipeline state (such as **Set Line Color**) is sent to the renderer for processing. All other output commands (such as **PolyLine 3D, Execute Structure, Label, etc**) are skipped. The traversal is flat meaning that the current pipeline attributes will not get "pushed" when a structure in the path is executed. However, the current path offset is bumped for each output command that is encountered during accumulate state. |

6.5.4. Render Network

Name:
PEXRenderNetwork

Request:
rdr_id : RENDERER_ID
drawable_id : DRAWABLE_ID
s_id : STRUCTURE_ID

Errors:
Renderer, Drawable, Structure, Match, Alloc, RendererState |

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*.

This functionality is equivalent to the following (pseudo-)requests:

PEXBeginRendering(*rdr_id*, *drawable_id*)
PEXRenderElements(*rdr_id*, *s_id*, all elements of *s_id*) |
PEXEndRendering(*rdr_id*, *True*)

7. Renderer Picking

The semantics of picking are identical to rendering except that during picking primitives are hit tested instead of converted to pixels. In other words, all primitives which are passed to the rendering pipeline (ie: they do not satisfy the invisibility filter and they are not culled backfaces) are eligible for picking. Picking uses the geometric definition of primitives so it is possible for hollow, empty and transparent polygons to be picked. However it is implementation dependent whether rendering effects which increase the area of the primitive (such as linewidth) are considered.

Two types of hit test traversals are supported: "pick one" and "pick all". A pick one traversal will return at most one primitive which satisfies the pick criteria. A "pick all" traversal will return all primitives which satisfy the pick criteria, up to a user specified maximum number of hits. For both types of traversals the spatial test which is performed on the primitives is controlled by the hit box which is specified in *pick_data*.

The hit box may be specified in device coordinates (if the server supports *DC_HitBox* pick devices) or in NPC space (if the server supports *NPC_HitVolume* pick devices). The supported pick device types can be inquired via **PEXGetEnumeratedTypeInfo**. A device coordinate hit box is specified by a *pick_position* and a *pick_distance*. The *pick_position* is the device coordinate center for the hit box and the shape of the hit box (ie: square, rectangle, circle) is implementation dependent. *pick_distance*, also measured in device coordinates, defines the half-width or radius of the hit box. The NPC hit volume is specified by two points in NPC space. Primitives that lie within or intersect the hit box are considered "hit".

The *pick_method* determines which "hit" primitives will be picked. For pick one, *pick_method* can be set to: *Last*, *ClosestZ*, *VisibleAny*, or *VisibleClosest*. For pick all, *pick_method* can be set to: *All* or *Visible*. An error will be returned if the specified *pick_method* is not supported on the connection. The supported pick one and pick all methods can be inquired using **PEXGetEnumeratedTypeInfo**. PEX servers are required to support *Last* for pick one and *All* for pick all.

The *pick_inclusion* and *pick_exclusion* renderer attributes specify the name sets to be used to filter picked primitives after a pick traversal. The pick requests will only return primitives which are not excluded by the pick filter. If possible, pick one will return a primitive which most closely satisfies the pick criteria and also passes the pick filter test. A flag is returned from pick one requests to indicate whether a primitive which did not satisfy the pick filter was a better candidate with the specified *pick_method*. The dynamics of *pick_inclusion* and *pick_exclusion* can be determined by calling **PEXGetRendererDynamics**.

During a pick one traversal at most one primitive is picked. If the *pick_method* is *Last* then the last "hit" primitive which satisfies the pick filter test is returned. If the *pick_method* is *ClosestZ* then the "hit" primitive which has a z value closest to the front clipping plane and satisfies the pick filter is returned. If multiple primitives satisfy this criteria then any of them may be returned. If the *pick_method* is *VisibleAny* then any "hit" primitive which is visible (taking the current HLHSR mode into account) and satisfies the pick filter may be returned. If the *pick_method* is *VisibleClosest* then all "hit" primitives which are visible are first selected. Of the selected primitives, the one which is closest to the *pick_position* and satisfies the pick filter is returned. If an NPC hit volume was specified then the *pick_position* is the center of the volume. The z buffer contents may not be preserved when using the "visible" pick methods.

The "visible" pick methods only return primitives which would be visible in the pick aperature if they were rendered using the renderer’s current HLHSR mode. For example, if the HLHSR mode is set to *ZBuffer*, the pick one method *ClosestZ* may return a primitive which is occluded by another primitive which does not satisfy the pick filter test. However, pick one method *VisibleAny* guarantees that the picked primitive is not occluded.

During a "pick all" traversal multiple primitives may be picked. If the *pick_method* is *All* then the first *max_hits* primitives, which lie within or intersect the hit box, are returned. If the *pick_method* is *Visible* then the first *max_hits* primitives which are visible (taking the current HLHSR mode into account) and lie within or intersect the hit box, are returned. Pick first can be accomplished by setting *max_hits* equal to one.

PickAll is further controlled by the *pick_start_path* renderer attribute. *pick_start_path* indicates where to begin the next *PickAll* traversal and it is bound at the start of a *PickAll* traversal. |

If the drawable that is associated with a renderer is destroyed or resized while the renderer is performing a pick one or pick all hit test, the pick operation is terminated and the renderer is returned to the *Idle* state. All subsequent output and traversal commands (e.g., **PEXRenderOutputCommands**, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**) are ignored, up to and including the final **PEXEndPickOne** or **PEXEndPickAll** request. No errors are generated but *pick_status* will be set to *Aborted* in the pick reply. |

If the drawable that is associated with a renderer is moved, exposed, or occluded while the renderer is performing a pick one or pick all hit test, the server must continue to process output commands using the newly-modified window hierarchy until the pick operation is explicitly or implicitly terminated. |

7.1. Pick One |

7.1.1. Begin Pick One |

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: |
Renderer, Drawable, Match, Alloc, RendererState, Value |

Description: |

This request starts a client side pick one traversal. When a renderer is in "pick one" mode, primitives will be hit tested instead of converted to pixels and, a hierarchical path to the picked primitive will be maintained. *pick_method* specifies the pick criteria to use so that only one primitive will be selected. It may be set to *Last*, *ClosestZ*, *VisibleAny*, or *VisibleClosest*. (**PEXBeginPickAll** will return the *First* picked primitive when *max_hits* is set to one.) If the specified *pick_method* is not supported then a *Value* error will be returned. |

The pick aperature is defined by *pick_data*. Either a *DC_HitBox* or *NPC_HitVolume* can be specified. |

s_id is the root of the client structure network. The current path is initialized to structure *s_id* and element offset zero before the pick traversal begins. The pick path which is returned from a pick one traversal is equivalent to the renderer's current path at the time the picked primitive was processed. |

If the specified drawable does not have the same root and depth as the drawable that was passed in to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRendererTargets**, a *Match* error will be generated. If the *renderer_state* is set to *Rendering* or *Picking*, then the operation will be aborted, the **PEXBeginPickOne** request will be executed, and a *RendererState* error returned. |

The existing PEX requests that process output commands or manipulate the current attributes (ie: **PEXRenderOutputCommands**, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**) can be issued when the renderer is in *Picking* mode. They will have the same semantics except that primitives are hit tested instead of converted to pixels. |

7.1.2. End Pick One

Name:

PEXEndPickOne

Request:

rdr_id : RENDERER_ID

Reply:

pick_status : PICK_STATUS
better_pick : BOOLEAN
pick_path : LISTofPICK_ELEMENT_REF

Errors:

Renderer, RendererState

Description:

This request places the renderer in *Idle* mode and returns the pick status along with the hierarchical pick path. If a primitive was picked, then *pick_status* will be set to *Ok* and *pick_path* will contain the renderer's current path at the time the picked primitive was processed. If no primitive was picked then *pick_status* will be set to *NoPick* and *pick_path* will be empty. If the renderer's drawable was destroyed or resized during the pick operation then *pick_status* will be set to *Aborted* and *pick_path* will be empty.

If there was a primitive which more closely satisfied the pick criteria but did not pass the pick filter test then *better_pick* will be set to *True*. Otherwise *better_pick* will be set to *False*.

If the *renderer_state* is *Idle* when this request is received (i.e., no picking is in progress or picking was aborted due to a resize), then this request is ignored but no error is generated. If the *renderer_state* is *Rendering* or a "pick all" is in progress then a *RendererState* error will be generated.

7.1.3. Pick One

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 : PICK_STATUS
better_pick : BOOLEAN
pick_path : LISTofPICK_ELEMENT_REF

Errors:

Renderer, Drawable, Structure, Match, Alloc, RendererState, Value

Description:

This request causes a pick one traversal to be performed on the structure network rooted at *s_id*. It returns the pick status along with the hierarchical pick path.

This request is equivalent to the following (pseudo-)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*) |

7.2. Pick All |

7.2.1. Begin Pick All |

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: |
Renderer, Drawable, Match, Alloc, RendererState, Value |

Description: |

This request starts a client side "pick all" traversal. When the renderer is in "pick all" mode, primitives are hit tested instead of converted to pixels. All picked primitives are recorded until *pick_max_hits* is reached. Additional primitives will not be recorded. Once the the maximum number of hits is reached, the server may choose to ignore all subsequent primitives. When *max_hits* is set to one this request will return the *First* picked primitive. |

pick_method specifies the pick criteria to use during the "pick all" traversal. It may be set to *All* or *Visible*. If the specified *pick_method* is not supported then a *Value* error will be returned. |

The pick aperature is defined by *pick_data*. Either a *DC_HitBox* or *NPC_HitVolume* can be specified. |

s_id is the root of the client structure network. The current path is initialized to structure *s_id* and element offset zero before the pick traversal begins. All pick paths which are returned from a "pick all" traversal are equivalent to the renderer's current path at the time the picked primitive was processed. |

If *send_event* is *True* and *pick_method* is *All*, then a **MaxHitsReached** event will be sent to the client whenever the maximum number of hits is reached by the server. (If the *pick_method* is *Visible* then a complete traversal must be performed before any primitives can be picked.) Upon receiving the event, the client should end the traversal and see if the recorded hits are sufficient. **PEXGetImpDepConstants** returns a flag indicating whether this event is supported by the server. |

If the specified drawable does not have the same root and depth as the drawable that was passed in to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRendererTargets**, a *Match* error will be generated. If the *renderer_state* is *Rendering* or *Picking*, then the operation will be aborted, the **PEXBeginPickAll** request will be executed, and a *RendererState* error returned. |

7.2.2. End Pick All

Name:

PEXEndPickAll

Request:

rdr_id : RENDERER_ID

Reply:

pick_status : PICK_STATUS
more_picks : PICK_ALL_STATE
pick_paths : LISTofLISTofPICK_ELEMENT_REF

Errors:

Renderer, RendererState

Description:

This request terminates the current client side "pick all" traversal, places the renderer in *Idle* mode, and returns the pick status along with the pick paths of any picked primitives. If one or more primitives were picked, then *pick_status* will be set to *Ok* and *pick_paths* will contain a hierarchical path for each picked primitive, stored in traversal order. The hierarchical pick path is equivalent to the renderer's current path at the time the picked primitive was processed. If no primitives were picked then *pick_status* will be set to *NoPick* and *pick_paths* will be empty. If the renderer's drawable was destroyed or resized during the pick operation then *pick_status* will be set to *Aborted* and *pick_paths* will be empty.

If all picks were recorded then *more_picks* will be set to *NoMoreHits* and the renderer's *pick_start_path* attribute will be empty. If the maximum number of hits was reached and additional picks were detected, then *more_hits* will be set to *MoreHits* and *pick_start_path* will be set to the last picked primitive which was recorded. If the server, after reaching the maximum number of hits, started ignoring all subsequent output commands, then *more_hits* will be set to *MayBeMoreHits* and *pick_start_path* will be set to the last element processed by the renderer before it started ignoring primitives.

If the *renderer_state* is *Idle* when this request is received (i.e., no picking is in progress or picking was aborted due to a resize), then this request is ignored but no error is generated. If the *renderer_state* is *Rendering* or a "pick one" is in progress then a *RendererState* error will be generated.

7.2.3. Pick All

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 : PICK_STATUS
more_picks : PICK_ALL_STATE
pick_paths : LISTofLISTofPICK_ELEMENT_REF

Errors:

Renderer, Drawable, Match, Alloc, RendererState, Value, Path

Description:

This request traverses the structure network specified in the renderer's *pick_start_path* attribute. Hit testing begins after the last element specified in *pick_start_path*. If the start path does not define a valid hierarchical path, a *Path* error is generated and no results are returned.

The pick aperature is defined by *pick_data*. Either a *DC_HitBox* or *NPC_HitVolume* can be specified.

The paths of all hit primitives are recorded until *pick_max_hits* is reached. These paths are returned in traversal order in *pick_paths*. Once the maximum number of paths is recorded, the server may choose to ignore subsequent primitives and return the results found. When *max_hits* is set to one this request will return the *First* picked primitive.

If all possible hits were recorded then *more_picks* will be set to *NoMoreHits* and the renderer's *pick_start_path* attribute will be empty. If the maximum number of hits was reached and additional hits were detected, then *more_hits* will be set to *MoreHits* and *pick_start_path* will be set to the last picked primitive which was recorded. If the server, after reaching the maximum number of hits, started ignoring all subsequent output commands, then *more_hits* will be set to *MayBeMoreHits* and *pick_start_path* will be set to the last element processed by the renderer before it started ignoring primitives.

This request is equivalent to the following (pseudo-)requests:

PEXBeginPickAll(*pick_method*, *send_event*, *rdr_id*, *drawable_id*, *s_id*, *pick_max_hits*, *pick_data*)
PEXRenderElements(*rdr_id*, *s_id*, all elements of *s_id*)
PEXEndPickAll(*rdr_id*)

8. Structures

This section details the usage and management of structure resources. A structure is a resource which stores output commands for later execution. Structures have two settable attributes and a number of attributes that can only be inquired. The modifiable attributes are:

element_ptr
: This attribute contains the offset of an element in the structure. Elements are numbered consecutively, starting at offset one. The element pointer determines the position in the structure at which the next editing or element query operation will occur. Whenever a structure is used as a destination in a structure editing request, its element pointer attribute will be updated as a side effect of the operation. When a structure is created, its element pointer is set to zero.

editing_mode
: The editing mode attribute specifies how editing operations will affect the structure. If the mode is *StructureInsert*, subsequent requests to create structure elements will cause elements to be inserted into the structure. Elements will be *inserted* into the structure *after* the structure element specified by the element pointer. The element pointer will then be incremented by the number of elements inserted. If the mode is *StructureReplace*, output requests used to create structure elements will cause structure elements to *replace* elements starting at the location specified by the element pointer. When a structure is created, its editing mode is set to *StructureInsert*.

A structure's non-modifiable attributes include its size (number of elements and length in four byte units if it were to be returned to the client) and the number of times it is referenced by other structures. Additional information about structures (ancestors, descendants, etc.) is also available and can be inquired separately.

8.1. Structure Resource Management

A structure is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation, deletion, and general manipulation of structures.

8.1.1. Create Structure

Name:
: **PEXCreateStructure**

Request:
: *s_id :* STRUCTURE_ID

Errors:
: IDChoice, Alloc

Description:
: This request creates a structure resource with the specified *s_id*.

8.1.2. Copy Structure

Name:

PEXCopyStructure

Request:

src_s_id : STRUCTURE_ID
dest_s_id : STRUCTURE_ID

Errors:

Structure

Description:

The structure elements in *src_s_id* are copied to *dest_s_id*. Any structure elements in *dest_s_id* are deleted prior to the copy operation. The destination structure must already exist as a valid resource. *Src_s_id*'s element pointer and editing mode are also copied to *dest_s_id*.

8.1.3. Destroy Structures

Name:

PEXDestroyStructures

Request:

list : LISTofSTRUCTURE_ID

Errors:

Structure

Description:

This request destroys each of the structure resources that is specified in *list* and removes all references to it in the server. Any structure elements that reference a structure in *list* will be destroyed, and any structure in *list* that is posted to a PHIGS workstation resource will be unposted. Any paths in search contexts or pick measures that contain the resource ID of any structure in *list* may still be queried. Such paths may still contain the resource ID of a structure after it has been destroyed. However, any attempts to perform a **PEXSearchNetwork** or a **PEXUpdatePickMeasure** using such a path will result in a *Path* error being generated.

Any structure that had structure elements removed due to a **PEXDestroyStructures** request will have their element pointer modified in the following way. If the element pointer is at a position before any of the elements that were deleted, the element pointer remains unchanged. If the element pointer was at the position of an element that was deleted, it will be set to the previous element. If the element pointer was after elements that were deleted, it will be updated so as to point at the same element it pointed to prior to the **PEXDestroyStructures** request.

8.2. Structure Inquiry

These requests inquire attributes and other information about structures.

8.2.1. Get Structure Info

Name:

PEXGetStructureInfo

Request:

fp_format : FLOAT_FORMAT
s_id : STRUCTURE_ID |
item_mask : BITMASK_SHORT

Reply:

editing_mode : EDIT_MODE
element_ptr : CARD32
num_elements : CARD32
total_length : CARD32
has_refs : BOOLEAN

Errors:

Structure, FloatingPointFormat, Value

Description:

This request returns information about the structure specified by *s_id*. *Item_mask* indicates which fields in the reply are required by the client. Items indicated by *item_mask* will have valid values returned, and the remaining values will be returned as undefined values. The current value of the structure's editing mode attribute will be returned in *editing_mode*. The current value of the structure's element pointer attribute will be returned in *element_ptr*. The number of structure elements in the specified structure will be returned in *num_elements*. If the structure is empty, *num_elements* will be zero. The total number of four-byte units necessary to store all elements of the structure (on the client side) will be returned in *total_length*. (This provides the client the information needed to allocate memory prior to doing a **PEXFetchElements** request.) If there are any "execute structure" elements in other structure resources that reference *s_id*, *has_refs* will be returned as *True,* otherwise it will be returned as *False*. The information returned will be computed as if the floating point type in *fp_format* were being used to return the data.

8.2.2. Get Element Info

Name:

PEXGetElementInfo

Request:

fp_format : FLOAT_FORMAT
s_id : STRUCTURE_ID
range : ELEMENT_RANGE

Reply:

info : LISTofELEMENT_INFO

Errors:

Structure, Value, FloatingPointFormat

Description:

This request returns information about elements in the structure specified by *s_id*. The offsets that define the range of elements about which information is to be obtained are computed using the positions in *range* in the following fashion. If *whence* is *Beginning*, the computed offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the structure. *Offset* can be a negative number.

If either computed offset is less than zero, it will be set to zero before obtaining the element information. If either computed offset is greater than the number of elements in the structure, it will be set to the offset of the last structure element in the structure. The request returns each element's ELEMENT_TYPE field, and the length of the element in units of four bytes. The sum of the lengths of all elements in the structure will be equal to the length that can be inquired with the **PEXGetStructureInfo** request. The information returned will be computed as if the floating point type in *fp_format* were being used to return the data. No information will be returned for inquiries on element offset zero. The element pointer attribute of *s_id* is not affected by this request.

8.2.3. Get Structures In Network

Name:

PEXGetStructuresInNetwork

Request:

s_id : STRUCTURE_ID
which : {*All, NoCrossRefs*}

Reply:

structures : LISTofSTRUCTURE_ID

Errors:

Structure, Value

Description:

This request returns a list of unique structure resource IDs that are referenced in the structure network rooted at *s_id*. If *which* is *All*, all of the structure resources referenced in the structure network rooted at *s_id* will be returned in the list *structures*. If *which* is *NoCrossRefs*, only the IDs of structures in the network that aren't referenced outside the specified structure network are returned. *S_id* will always be returned in the list, unless *s_id* is an invalid structure ID, in which case the returned list will be empty.

8.2.4. Get Ancestors

Name:

PEXGetAncestors

Request:

s_id : STRUCTURE_ID
path_part : {*TopPart, BottomPart*}
path_depth : CARD32

Reply:

paths : LISTofLISTofELEMENT_REF

Errors:

Structure, 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 will be returned. If *path_part* = *BottomPart*, only the last *path_depth* structure id/element offset pairs in each path will be returned. Only unique paths will be returned (i.e., there will be no duplicates in the list of returned paths).

For instance, specifying *path_depth* = 0 and *path_part* = *TopPart* would cause all paths leading to *s_id* to be returned. Specifying *path_depth* = 1 and *path_part* = *TopPart* would cause all the structures at the top of each structure network containing *s_id* to be returned. Specifying *path_depth* = 1 and *path_part* = *BottomPart* would allow you to determine whether or not *s_id* is referenced. Specifying *path_depth* = 2 and *path_part* = *BottomPart* would allow you to determine the number of immediate ancestors for *s_id* as well as returning their resource IDs.

8.2.5. Get Descendants

Name:

PEXGetDescendants

Request:

s_id : STRUCTURE_ID
path_part : {*TopPart, BottomPart*}
path_depth : CARD32

Reply:

paths : LISTofLISTofELEMENT_REF

Errors:

Structure, 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 will be returned. If *path_part* = *BottomPart*, only the last *path_depth* structure id/element offset pairs in each path will be returned. Only unique paths will be returned (i.e., there will be no duplicates in the list of returned paths).

For instance, specifying *path_depth* = 0 and *path_part* = *TopPart* would cause all paths leading from *s_id* to leaf nodes to be returned. Specifying *path_depth* = 1 and *path_part* = *TopPart* would cause 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*.

8.2.6. Fetch Elements

Name:

PEXFetchElements

Request:

fp_format : FLOAT_FORMAT
s_id : STRUCTURE_ID
range : ELEMENT_RANGE

Reply:

elements : LISTofOUTPUT_CMD

Errors:

Structure, FloatingPointFormat, Value

Description:

This request returns a range of structure elements in the structure specified by *s_id*. The offsets that define the range of elements to be fetched are computed using the positions in *range* in the following fashion. If *whence* is *Beginning*, the computed offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the structure. *Offset* can be a negative number.

If either computed offset is less than zero, it will be set to zero before the fetch occurs. If either computed offset is greater than the number of elements in the source structure, it will be set to the offset of the last structure element in the source structure. The element pointer attribute of *s_id* is not affected by the fetch operation. The data for each structure element will be returned in the same format as described Section 3 - *Output Commands* section.

Floating point values in *elements* will be returned in the floating point format specified in *fp_format* (if it is supported by the PEX implementation).

No information will be returned for inquiries on element offset zero.

8.3. Structure Resource Attribute Modification

This section contains requests that can be used to modify structure resource attributes.

8.3.1. Set Editing Mode

Name:

PEXSetEditingMode

Request:

s_id : STRUCTURE_ID
mode : EDIT_MODE

Errors:

Structure, Value

Description:

This request modifies *s_id*'s editing mode attribute. The editing mode attribute specifies how editing operations will affect the structure. If the editing mode is set to *StructureInsert*, subsequent requests to create structure elements will cause elements to be inserted into the structure. Elements will be *inserted* into the structure *after* the structure element specified by the element pointer. The element pointer will then be incremented by the number of elements inserted. If the mode is *StructureReplace*, output requests used to create structure elements will cause structure elements to *replace* elements starting at the location specified by the element pointer.

8.3.2. Set Element Pointer

Name:

PEXSetElementPointer

Request:

s_id : STRUCTURE_ID
position : ELEMENT_POS

Errors:

Structure, Value

Description:

This request sets the element pointer attribute of the structure specified by *s_id*. The element pointer attribute of the structure will be set to the position specified by *position*, which contains a *whence* and *offset* pair. If *whence* is *Beginning*, the element pointer is set to the specified *offset*. If *whence* is *Current*, the new element pointer value is computed by adding *offset* to the current value of the element pointer. If *whence* is *End*, the new element pointer value is computed by adding the specified *offset* to the offset of the last element in the structure. *Offset* can be a negative number.

If the resultant value of the element pointer is less than zero, the element pointer is set to zero. If the resultant value of the element pointer is greater than the number of elements in the structure, then the element pointer is set to the offset of the last element in the structure.

8.3.3. Set Element Pointer At Label

Name:

PEXSetElementPointerAtLabel

Request:

s_id : STRUCTURE_ID
label : INT32
offset : INT32

Errors:

Structure, Label

Description:

This request sets the element pointer attribute of the structure specified by *s_id*. A search is conducted for the next occurrence of a "label" structure element containing *label*. The search for the label starts at the current element pointer plus one, and proceeds in the forward direction. If a "label" structure element containing *label* is found, the element pointer for the structure is set to the offset of the located label plus *offset*.

If the resultant value of the element pointer is less than zero, the element pointer is set to zero. If the resultant value of the element pointer is greater than the number of elements in the structure, then the element pointer is set to the offset of the last element in the structure. This is a non-descending search (i.e., the search does not include any structures referenced by "execute structure" elements). If no occurrence of the specified label is found, an error is generated and the structure's element pointer is left unchanged.

8.3.4. Element Search

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:

Structure, Value

Description:

This request conducts a search for the first occurrence of the specified element type in the structure specified by *s_id*. The offset at which to begin searching is computed using the *whence* and *offset* found in *position*. If *whence* is *Beginning*, the search will begin at the element specified by *offset*. If *whence* is *Current*, the specified *offset* is added to the current value of *s_id*'s element pointer to determine the element at which to begin the search. If *whence* is *End*, the element at which to begin the search is computed by adding the specified *offset* to the offset of the last element in the structure. *Offset* can be a negative number.

If the computed offset is less than zero, the search will begin at the position preceding the first element in the structure. If the computed offset is greater than the number of elements in the structure, then the search will begin at the last element in the structure. The search always includes the starting element.

An element will be selected if its element type is not contained in *excl* and its element type is included in *incl*. An element type of *All* causes all element types to match. If a structure element type is in both the inclusion and exclusion sets, it will be excluded.

The search terminates if a match is found or if the limits of the structure are reached. The search progresses from the start point and proceeds either forward in the structure or backward, depending on *direction*. This is a non-descending search (i.e., the search does not include any structures referenced by "execute structure" elements). If the search succeeds in finding a match, a status of *Found* is returned, and the offset of the matching element is returned in *found_offset*. If the search is unsuccessful, a status of *NotFound* is returned, and a value of zero is returned in *found_offset*.

The element pointer attribute of *s_id* is not modified as a result of this request.

8.4. Structure Editing

This section contains requests that can be used to modify structure resources.

8.4.1. Store Elements

Name:

PEXStoreElements

Request:

fp_format : FLOAT_FORMAT
s_id : STRUCTURE_ID
elements : LISTofOUTPUT_CMD

Errors:

Structure, FloatingPointFormat, OutputCommand

Description:

This request causes elements to be stored in the structure whose resource ID is specified by *s_id*. If the *editing_mode* attribute of the structure is set to *StructureInsert*, each entry in *elements* is used to create a structure element in the structure after the element specified by the structure's *element_ptr* attribute. Output commands in the request are processed until one is found to be in error, or until the entire list has been processed. The erroneous output command and all others following it in the request are ignored (they will *not* be added to the structure), and an *OutputCommand* error is returned to the client. The element pointer is incremented by one for each such insertion. Therefore, at the conclusion of the request, the element pointer is left pointing at the last inserted element.

If the *editing_mode* attribute is set to *StructureReplace*, a range of elements is first deleted from the structure as if **PEXDeleteElements** were called with a *position1* whence/offset of {*Current*, 0} and a *position2* whence/offset of {*Current*, $n - 1$} where n is the number of elements to be deleted. (If element zero is included as part of the range to be deleted, all elements in the range except for element zero will be deleted.) If there are fewer elements between the structure's element pointer and the end of the structure than there are output commands in *elements*, then only the elements between the element pointer and the end of the structure will be deleted. The element pointer will be left pointing at the element preceding the deleted range. After the deletion, the output commands specified in *elements* are inserted at the element pointer position as described above.

Floating point values in *elements* will be in the floating point format specified in *fp_format*. The formats for the various types of output commands can be found in the "Output Commands" section.

8.4.2. Delete Elements

Name:

PEXDeleteElements

Request:

s_id : STRUCTURE_ID
range : ELEMENT_RANGE

Errors:

Structure, Value

Description:

This request deletes elements in *range* from the structure whose ID is specified in *s_id*. The range of elements to be deleted is computed using the *whence* and *offset* values of *position1* and *position2*.

The offsets that define the range of elements to be deleted are each computed using the positions in *range* in the following fashion. If *whence* is *Beginning*, the computed offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the structure. *Offset* can be a negative number.

If either computed offset is less than zero, it will be set to zero before the deletion occurs. If either computed offset is greater than the number of elements in the structure, it will be set to the offset of the last structure element. It is not necessary for the first offset to be less than the second. The deletion is inclusive, meaning that the elements at the boundary of the deletion range are also deleted. It does not matter which of the two offsets determines the lower end of the range to be deleted and which determines the higher end of the range. Deleting element zero is effectively a noop. After the deletion operation, the structure's element pointer will be set to the element preceding the range of deleted elements.

8.4.3. Delete Elements To Label

Name:

PEXDeleteElementsToLabel

Request:

s_id : STRUCTURE_ID
position : ELEMENT_POS
label : INT32

Errors:

Structure, Label, Value

Description:

This request deletes all elements between a computed offset and a specified *label* in the structure whose ID is specified in *s_id*. The offset is computed using *position* in the following fashion. If *whence* is *Beginning*, the computed offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the structure. *Offset* can be a negative number.

If the computed offset is less than zero, it will be set to zero before the deletion occurs. If the computed offset is greater than the number of elements in the structure, it will be set to the offset of the last structure element. Elements are deleted starting at the element immediately after the computed offset up to the next occurrence of the *label*. The label itself is not deleted. The structure's element pointer is set to point at the element immediately preceding the range of deleted elements. If the specified label is not found, no deletion occurs and the structure's element pointer is left unchanged.

8.4.4. Delete Elements Between Labels

Name:

PEXDeleteElementsBetweenLabels

Request:

s_id : STRUCTURE_ID
label1 : INT32
label2 : INT32

Errors:

Structure, Label

Description:

This request deletes all elements between *label1* and *label2* in the structure whose ID is specified in *s_id*. A search for *label1* is first performed starting at the element whose offset is specified by the structure's element pointer. A search for *label2* is then performed, starting at the element after *label1*. The range of elements between the two labels is then deleted. The two label elements themselves are not deleted. The structure's element pointer is set to point at the element immediately preceding the range of deleted elements (the element containing *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.

8.4.5. Copy Elements

Name:

PEXCopyElements

Request:

src_s_id : STRUCTURE_ID
src_range : ELEMENT_RANGE
dest_s_id : STRUCTURE_ID
dest_position : ELEMENT_POS

Errors:

Structure, Value

Description:

This request copies elements in *src_range* from the structure specified in *src_s_id* to the *dest_position* in the structure specified in *dest_s_id*. The range of elements to be copied is computed using the *whence* and *offset* values of *position1* and *position2* in *src_range*.

The offsets that define the range of elements to be copied are each computed using the positions in *src_range* in the following fashion. If *whence* is *Beginning*, the computed offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *src_s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the structure. *Offset* can be a negative number.

If either computed offset is less than zero, it will be set to zero before the copy occurs. If either computed offset is greater than the number of elements in the source structure, it will be set to the offset of the last structure element in the source structure. It does not matter which of the two offsets determines the lower end of the range to be copied and which determines the higher end of the range (i.e., the element order will not be reversed if *offset1* is greater than *offset2*).

The position to which the elements will be copied is computed using *dest_position* in the following fashion. If *whence* is *Beginning*, the computed destination offset is just the value specified by *offset*. If *whence* is *Current*, the offset is computed by adding *offset* to *dest_s_id*'s element pointer. If *whence* is *End*, the offset is computed by adding *offset* to the number of elements in the destination structure. *Offset* can be a negative number.

It is permissible for *src_s_id* and *dest_s_id* to be the same. If this is the case, the copy operation performs as if the indicated range is copied to a temporary location, and then inserted relative to the destination position.

After the copy operation, the destination structure's element pointer is updated to point at the last element inserted in the destination structure. The editing mode attribute of *dest_s_id* is ignored during this request; copied elements are always inserted into the destination structure, never used to replace existing structure elements.

8.4.6. Change Structure References

Name:

PEXChangeStructureReferences

Request:

old_s_id : STRUCTURE_ID
new_s_id : STRUCTURE_ID

Errors:

Structure

Description:

This request changes any "execute structure" elements in the server that reference the structure specified by *old_s_id* into "execute structure" elements that reference the structure specified by *new_s_id*.

Any references to the structure *new_s_id* that existed before this request are not affected. On all PHIGS workstation resources where *new_s_id* is already 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. If there were references to the structure resource specified by *old_s_id* and the structure resource specified by *new_s_id* does not exist, an error is returned and no action is taken.

9. Name Sets

A *name set* is a PEX resource that is used to filter output commands during rendering, picking, and searching operations. Name sets typically make fairly small demand on server memory resources, but they are of variable size so this can vary depending on their contents.

9.1. Name Set Resource Management

The name set is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation, freeing, and copying of name set resources.

9.1.1. Create Name Set

Name:

PEXCreateNameSet

Request:

ns_id : NAME_SET_ID

Errors:

IDChoice, Alloc

Description:

This request creates a name set resource for the specified *ns_id*. The name set is initialized to an empty list when the resource is created.

9.1.2. Copy Name Set

Name:

PEXCopyNameSet

Request:

src_ns_id : NAME_SET_ID
dest_ns_id : NAME_SET_ID

Errors:

NameSet

Description:

This request copies the source name set *src_ns_id* to a destination name set *dest_ns_id* after first emptying the contents of *dest_ns_id*. The *dest_ns_id* must already exist as a valid resource.

9.1.3. Free Name Set

Name:

PEXFreeNameSet

Request:

ns_id : NAME_SET_ID

Errors:

NameSet

Description:

This request deletes the association between the resource ID and the name set. The name set storage will be freed when no other resource references it.

9.2. Name Set Inquiry

These requests inquire the contents of name set resources.

9.2.1. Get Name Set

Name:

PEXGetNameSet

Request:

ns_id : NAME_SET_ID

Reply:

names : LISTofNAME

Errors:

NameSet

Description:

This request will return the contents of the name set specified by *ns_id*.

9.3. Name Set Modification

These requests can be used to modify the contents of name set resources.

9.3.1. Change Name Set

Name:

PEXChangeNameSet

Request:

ns_id : NAME_SET_ID
action : {*Replace, Add, Remove*}
names : LISTofNAME

Errors:

NameSet, Value

Description:

This request changes the contents of a name set resource. If *action* is *Add*, the specified list of names is added to the name set. If *action* is *Remove*, the specified list of names is removed from the name set. If *action* is *Replace*, all the names in the name set are removed, then the specified list of names is added to the name set. If an attempt is made to remove a name which is not in the nameset, then the name will be ignored and no error will be generated. |

10. Search Contexts

A *search context* is a PEX resource that allows clients to perform searching operations on structure networks. This section describes the operations that can be performed on search context resources and the operations that can be performed using search context resources. PEX currently provides no support for searching of client-side structure networks.

Some of the requests in this section affect attributes of a search context. The *item_mask* and *item_list* parameters specify which components are to be affected. Each bit in the *item_mask* indicates whether or not the corresponding attribute is affected. In the cases where search context attributes are being set or queried, there is a corresponding entry in the *item_list* for each set bit in *item_mask*. It is therefore possible to affect one or many search context attributes with a single request.

Two of the search context attributes contain name set resource IDs. If a name set is created, bound to a search context, and then freed, the contents of the name set will remain, since it is still being referenced by the search context. However, when a search context 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.

When a text primitive 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.†

The search context components, in order, are listed in the following table.

Attribute Name	**Data Type**	**Default Value**
search_pos	COORD_3D	(0.0,0.0,0.0)
search_dist	FLOAT	0.0
search_ceiling	CARD16	1
model_clip_flag	BOOLEAN	*False*
start_path	LISTofELEMENT_REF	*Null*
normal_list	LISTofNAME_SET_PAIR	*Null*
inverted_list	LISTofNAME_SET_PAIR	*Null*

The attributes of the search context resource are defined as follows:

search_pos

This attribute specifies the search reference position in world coordinates.

search_dist

This attribute specifies a distance from the search reference position in world coordinates. A successful search occurs only when an output primitive element is found that satisfies the search filter criteria *and* is within the specified distance from the search reference point. In order to satisfy the search criteria if *search_dist* is less than or equal to zero, the primitive must intersect the *search_pos*.

search_ceiling

This attribute defines the ceiling of the search operation. The search ceiling is an index into the list contained in *start_path* and searching stops when the end of the structure specified by *search_ceiling* is reached. For example, if there are *n* items in *start_path*, and *search_ceiling* is set to *n*, the search traversal will terminate at the end of the last structure in the list. On the other hand, if *search_ceiling* is equal to one, then the search

† The criteria of closeness to the origin matches PHIGS semantics for incremental spatial search on annotation text. However, PHIGS specifies that the enclosing rectangle of text primitives be used when searching. The traversal-time values of the geometric text attributes, together with text font 1, text precision *Stroke*, character expansion factor 1 and character spacing 0, determine the spatial extent of the enclosing rectangle. At this time, PEX does not follow the PHIGS semantics for searching on text primitives. In order to release the current PEX documents in a timely manner, this issue has been deferred until a future version of PEX.

will effectively operate without a ceiling, terminating at the end of the first structure in *start_path*.

model_clip_flag

This attribute specifies whether modeling clipping must be performed during the search operation. If *True*, modeling clipping is performed using the modeling clipping attributes as they are encountered during the traversal. If *False*, no modeling clipping is performed and modeling clipping attributes that are encountered during the traversal are effectively ignored.

start_path

This attribute defines the structure network path that is to be used as the starting point for the search. Searching begins at the element *following* the one indicated by the start path.

normal_list

This attribute contains the list of name set resource ID pairs to be used as filters in the search operation. If the *normal_list* is *Null*, all name sets are considered accepted by the normal filter list.

inverted_list

This attribute contains the list of name set resource ID pairs to be inverted and used as filters in the search operation. If the *inverted_list* is *Null*, all name sets are considered accepted by the inverted filter list.

10.1. Search Context Resource Management

The search context is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation, freeing, and copying of search contexts.

10.1.1. Create Search Context

Name:

PEXCreateSearchContext

Request:

fp_format : FLOAT_FORMAT
sc_id : SEARCH_CONTEXT_ID
item_mask : BITMASK
item_list : LISTofVALUE

Errors:

IDChoice, Value, FloatingPointFormat, Alloc, Path, NameSet

Description:

This request creates a search context resource for the specified *sc_id*. The *item_mask* defines those search context attributes that are to be explicitly set at the time the resource is created. The *item_list* contains the corresponding list of values used to modify the newly-created search context. Floating point values in *item_list* will be in the floating point format specified in *fp_format*.

10.1.2. Copy Search Context

Name:

PEXCopySearchContext

Request:

src_sc_id : SEARCH_CONTEXT_ID
dest_sc_id : SEARCH_CONTEXT_ID
item_mask : BITMASK

Errors:

SearchContext, Value

Description:

This request copies the source search context *src_sc_id* to a destination search context *dest_sc_id*. The *dest_sc_id* must already exist as a valid resource. The *item_mask* indicates which values in the search context will be copied.

10.1.3. Free Search Context

Name:

PEXFreeSearchContext

Request:

sc_id : SEARCH_CONTEXT_ID

Errors:

SearchContext

Description:

This request deletes the specified search context resource and frees the storage associated with it.

10.2. Search Context Inquiry

The requests in this section can be used to inquire search context attributes.

10.2.1. Get Search Context

Name:

PEXGetSearchContext

Request:

fp_format : FLOAT_FORMAT
sc_id : SEARCH_CONTEXT_ID
item_mask : BITMASK

Reply:

item_list : LISTofVALUE

Errors:

SearchContext, FloatingPointFormat, Value

Description:

This request will return components of the search context specified by *sc_id*. The *item_mask* specifies which components are to be inquired and returned. The specified attributes of the search context will be returned in *item_list*. Floating point values in *item_list* will be returned in the floating point format specified in *fp_format*.

10.3. Search Context Modification

The requests in this section can be used to modify attributes of search context resources.

10.3.1. Change Search Context

Name:

PEXChangeSearchContext

Request:

fp_format : FLOAT_FORMAT
sc_id : SEARCH_CONTEXT_ID
item_mask : BITMASK
item_list : LISTofVALUE

Errors:

SearchContext, Value, FloatingPointFormat, Path, NameSet

Description:

This request changes components of a search context. The *item_mask* and *item_list* specify which components are to be changed. Each bit in the *item_mask* indicates whether or not there is a corresponding entry in the *item_list*. It is therefore possible to modify one or many search context attributes with a **PEXChangeSearchContext** request. Floating point values in *item_list* will be in the floating point format specified in *fp_format*.

10.4. Structure Network Searching

This section describes requests that use search context resources to perform structure network searching operations.

10.4.1. Search Network

Name:

PEXSearchNetwork

Request:

sc_id : SEARCH_CONTEXT_ID

Reply:

found_path : LISTofELEMENT_REF

Errors:

SearchContext, Path

Description:

This request causes a spatial search in world coordinates to be performed on a structure network. The parameters of the searching operation are found in the search context *sc_id*. The search begins at the element *following* the one indicated by the *start_path* attribute of *sc_id*. The search is terminated once 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*. If no element is found, *found_path* will be null. After the search has completed, the *start_path* attribute of *sc_id* will be set to the value that is returned in *found_path*.

For a structure element to be considered a candidate for the search, the current name set has to be accepted by all the name set pairs in the search context's *normal_list*, and has to be rejected by all of the name set pairs in the *inverted_list*. If the *normal_list* is *Null*, all possible name sets are accepted by the normal list. If the *inverted_list* is *Null*, all possible name sets are accepted by the inverted list. Therefore the default case is that all output primitives are considered.

11. PHIGS Workstations

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 the next section, "Picking".

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.

11.1. PHIGS Workstation Resource Management

The PHIGS workstation is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the creation, freeing, and copying of PHIGS workstation resources.

11.1.1. Create PHIGS Workstation

Name:

PEXCreatePhigsWKS

Request:

wks_id : PHIGS_WKS_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 : BUFFER_MODE

Errors:

IDChoice, Drawable, Match, LookupTable, NameSet, Alloc, Value

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. If *drawable_id* is a pixmap, then no double buffering will be performed.

11.1.2. Free PHIGS Workstation

Name:

PEXFreePhigsWKS

Request:

wks_id : PHIGS_WKS_ID

Errors:

PhigsWKS

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.

11.2. PHIGS Workstation Inquiry

This section defines requests that can be used to get information about a PHIGS workstation resource.

11.2.1. Get PHIGS Workstation Info

Name:

PEXGetWKSInfo

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
item_mask : WKS_BITMASK

Reply:

item_list : LISTofVALUE

Errors:

PhigsWKS, FloatingPointFormat, Value

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*.

In the table below, 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 attributes that may be returned, in order, are:

Attribute Name	Data Type	Default
display_update	DISPLAY_UPDATE	*VisualizeEach*
visual_state	VISUAL_STATE	*Correct*
display_surface	DISPLAY_STATE	*Empty*
view_update	UPDATE_STATE	*NotPending*
defined_views	LISTofTABLE_INDEX	View 0 defined
wks_update	UPDATE_STATE	*NotPending*
req_NPC_subvolume	NPC_SUBVOLUME	(0.0,0.0,0.0),(1.0,1.0,1.0)
cur_NPC_subvolume	NPC_SUBVOLUME	(0.0,0.0,0.0),(1.0,1.0,1.0)
req_wks_viewpt	VIEWPORT	{imp. dep., imp. dep., True}
cur_wks_viewpt	VIEWPORT	{imp. dep., imp. dep., True}
HLHSR_update	UPDATE_STATE	*NotPending*
req_HLHSR_mode	HLHSR_MODE	1
cur_HLHSR_mode	HLHSR_MODE	1
drawable_id	DRAWABLE_ID	create wks req.
marker_bundle	LOOKUP_TABLE_ID	create wks req.
text_bundle	LOOKUP_TABLE_ID	create wks req.
line_bundle	LOOKUP_TABLE_ID	create wks req.
interior_bundle	LOOKUP_TABLE_ID	create wks req.
edge_bundle	LOOKUP_TABLE_ID	create wks req.
color_table	LOOKUP_TABLE_ID	create wks req.
depth_cue_table	LOOKUP_TABLE_ID	create wks req.
light_table	LOOKUP_TABLE_ID	create wks req.
color_approx_table	LOOKUP_TABLE_ID	create wks req.

pattern_table	LOOKUP_TABLE_ID	create wks req.
text_font_table	LOOKUP_TABLE_ID	create wks req.
highlight_inclusion	NAME_SET_ID	create wks req.
highlight_exclusion	NAME_SET_ID	create wks req.
invisibility_inclusion	NAME_SET_ID	create wks req.
invisibility_exclusion	NAME_SET_ID	create wks req.
posted_structs	LISTofSTRUCTURE_INFO	*Null*
num_priorities	CARD32	imp. dep.
buffer_update	UPDATE_STATE	*NotPending*
req_buffer_mode	BUFFER_MODE	create wks req.
cur_buffer_mode	BUFFER_MODE	create wks req.

display_update
Returns the PHIGS workstation's current display update mode. This mode specifies how the PHIGS workstation attempts to visualize changes, and can be set with the **PEXSetDisplayUpdateMode** request.

visual_state
Returns 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.

display_surface
Returns 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.

view_update
Returns *Pending* or *NotPending*, depending on whether there are view modification requests that have yet to be made current.

defined_views
Returns 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.

wks_update
Returns *Pending* or *NotPending*, depending on whether there are workstation transformation requests that have yet to be made current.

req_NPC_subvolume
Returns the value for the PHIGS workstation's NPC subvolume specification that has been requested but has not yet been made current.

cur_NPC_subvolume
Returns the current value for the PHIGS workstation's NPC subvolume specification.

req_wks_viewpt
Returns the value for the PHIGS workstation's viewport specification that has been requested but has not yet been made current.

cur_wks_viewpt
Returns the current value for the PHIGS workstation's viewport specification.

HLHSR_update
Returns *Pending* or *NotPending*, depending on whether there are HLHSR mode requests that have yet to be made current.

req_HLHSR_mode
Returns the value for the PHIGS workstation's HLHSR mode that has been requested but has not yet been made current.

cur_HLHSR_mode
Returns the current value for the PHIGS workstation's HLHSR mode.

drawable_id
Returns the resource ID of the drawable that is being used as the display surface.

marker_bundle
Returns the resource ID of the PHIGS workstation's marker bundle table.

text_bundle
Returns the resource ID of the PHIGS workstation's text bundle table.

line_bundle
Returns the resource ID of the PHIGS workstation's line bundle table.

interior_bundle
Returns the resource ID of the PHIGS workstation's interior bundle table.

edge_bundle
Returns the resource ID of the PHIGS workstation's edge bundle table.

color_table
Returns the resource ID of the color lookup table that will be used by the PHIGS workstation to dereference indexed color values.

depth_cue_table
Returns the resource ID of the PHIGS workstation's depth-cue table.

light_table
Returns the resource ID of the PHIGS workstation's light table.

color_approx_table
Returns the resource ID of the color approximation table that will be used by the PHIGS workstation.

pattern_table
Returns the resource ID of the PHIGS workstation's pattern table.

text_font_table
Returns the resource ID of the PHIGS workstation's text font table.

highlight_inclusion
Returns the resource ID of the PHIGS workstation's highlight inclusion name set.

highlight_exclusion
Returns the resource ID of the PHIGS workstation's highlight exclusion name set.

invisibility_inclusion
Returns the resource ID of the PHIGS workstation's invisibility inclusion name set.

invisibility_exclusion
Returns the resource ID of the PHIGS workstation's invisibility exclusion name set.

posted_structs
Returns 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.

num_priorities
Returns the number of display priorities that are supported. A value of zero indicates that a continuous range of priorities is supported.

buffer_update
Returns *Pending* or *NotPending*, depending on whether there are buffer mode requests that have yet to be made current.

req_buffer_mode
Returns the value for the PHIGS workstation's buffer mode that has been requested but has not yet been made current.

cur_buffer_mode
Returns 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.

11.2.2. Get Dynamics

Name:

PEXGetDynamics

Request:

drawable_id : DRAWABLE_ID

Reply:

view_rep : DYNAMIC_TYPE
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, Match |

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 list of dynamics, in order, is as follows:

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 set
invisibility_filter	Changes to invisibility name set
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*.

11.2.3. Get View Representation

Name:

PEXGetViewRep

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
index : TABLE_INDEX

Reply:

view_update : {Pending, NotPending}
requested : VIEW_REP
current : VIEW_REP

Errors:

PhigsWKS, FloatingPointFormat, Value

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*.

11.3. PHIGS Workstation Manipulation

This section contains requests that modify PHIGS workstation resources.

11.3.1. Redraw All Structures

Name:

PEXRedrawAllStructures

Request:

wks_id : PHIGS_WKS_ID

Errors:

PhigsWKS

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*.

11.3.2. Update Workstation

Name:

PEXUpdateWorkstation

Request:

wks_id : PHIGS_WKS_ID

Errors:

PhigsWKS

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*.

11.3.3. Redraw Clip Region

Name:

PEXRedrawClipRegion

Request:

wks_id : PHIGS_WKS_ID
clip_list : LISTofDEVICE_RECT

Errors:

PhigsWKS

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.

11.3.4. Execute Deferred Actions

Name:

PEXExecuteDeferredActions

Request:

wks_id : PHIGS_WKS_ID

Errors:

PhigsWKS

Description:

This request causes all the deferred actions on the PHIGS workstation specified by *wks_id* to be executed.

11.3.5. Set View Priority

Name:

PEXSetViewPriority

Request:

wks_id : PHIGS_WKS_ID
index1 : TABLE_INDEX
index2 : TABLE_INDEX
priority : {*Higher*, *Lower*}

Errors:

PhigsWKS, Value

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.

11.3.6. Set Display Update Mode

Name:

PEXSetDisplayUpdateMode

Request:

wks_id : PHIGS_WKS_ID
display_update : DISPLAY_UPDATE

Errors:

PhigsWKS, Value

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 the "Extension Information" section that describes enumerated types.

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. *

11.3.7. Map DC to WC

Name:

PEXMapDCtoWC

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
dc_points : LISTofDEVICE_COORD

Reply:

wc_points : LISTofCOORD_3D
view_index : TABLE_INDEX

Errors:

PhigsWKS, FloatingPointFormat

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.

11.3.8. Map WC to DC

Name:

PEXMapWCtoDC

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
wc_points : LISTofCOORD_3D
view_index : TABLE_INDEX

Reply:

dc_points : LISTofDEVICE_COORD

Errors:

PhigsWKS, FloatingPointFormat

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.

11.4. PHIGS Workstation Update

This section defines requests that can be used to set "requested" values for PHIGS workstation resources.

11.4.1. Set View Representation

Name:

PEXSetViewRep

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
view_rep : VIEW_REP

Errors:

PhigsWKS, FloatingPointFormat, Alloc

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.

11.4.2. Set Workstation Window

Name:

PEXSetWKSWindow

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
NPC_subvolume : NPC_SUBVOLUME

Errors:

PhigsWKS, FloatingPointFormat

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.

11.4.3. Set Workstation Viewport

Name:

PEXSetWKSViewport

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
viewport : VIEWPORT

Errors:

PhigsWKS, FloatingPointFormat, Value

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.

11.4.4. Set HLHSR Mode

Name:

PEXSetHLHSRMode

Request:

wks_id : PHIGS_WKS_ID
mode : HLHSR_MODE

Errors:

PhigsWKS, Value, Alloc |

Description:

This request sets the *req_HLHSR_mode* of the PHIGS workstation specified by *wks_id* to the values 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.

11.4.5. Set Buffer Mode

Name:

PEXSetWKSBufferMode

Request:

wks_id : PHIGS_WKS_ID
buffer_mode : BUFFER_MODE

Errors:

PhigsWKS, Value, Alloc

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). An *Alloc* error will be returned if the requested mode is *Double* and the server cannot allocate any more image buffers.

11.5. Posting/Unposting Structures

This section describes the requests that can be used to post and unpost structures on a PHIGS workstation resource.

11.5.1. Post Structure

Name:

PEXPostStructure

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
s_id : STRUCTURE_ID
priority : FLOAT

Errors:

PhigsWKS, Structure, FloatingPointFormat

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.

11.5.2. Unpost Structure

Name:

PEXUnpostStructure

Request:

wks_id : PHIGS_WKS_ID
s_id : STRUCTURE_ID

Errors:

PhigsWKS, Structure

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.

11.5.3. Unpost All Structures

Name:

PEXUnpostAllStructures

Request:

wks_id : PHIGS_WKS_ID

Errors:

PhigsWKS

Description:

This request removes all structures from *wks_id*'s posted structure list.

11.5.4. Get PHIGS Workstation Postings

Name:

PEXGetWKSPostings

Request:

s_id : STRUCTURE_ID

Reply:

wks_id : LISTofPHIGS_WKS_ID

Errors:

Structure

Description:

This request returns a list of all of the PHIGS workstation resources to which *s_id* has been posted.

12. Workstation Picking |

The discussion of workstation picking includes *pick device descriptors* which are found in PHIGS workstation resources and *pick measures*. |

12.1. 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 the following table. 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.

Attribute Name	Data Type	Default Value
pick_status	{*NoPick, Ok*}	*NoPick*
pick_path	LISTofPICK_ELEMENT_REF	*Null*
pick_path_order	{*TopFirst, BottomFirst*}	*TopFirst*
pick_inclusion	NAME_SET_ID	*Null*
pick_exclusion	NAME_SET_ID	*Null*
pick_data_rec	LISTofCARD8	pick dev. dep.
prompt_echo_type	PROMPT_ECHO_TYPE	pick dev. dep.
echo_volume	VIEWPORT	{imp. dep., imp. dep., True}
echo_switch	{*NoEcho, Echo*}	*NoEcho*

The components of the pick device descriptor are defined as follows:

pick_status
: This attribute contains the initial pick status that will be bound to a pick measure resource.

pick_path
: This attribute contains the initial pick path that will be bound to a pick measure resource.

pick_path_order
: This attribute 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.

pick_inclusion
: This attribute specifies the resource ID of the name set resource that is to be used as the pick inclusion filter during picking operations.

pick_exclusion
: This attribute specifies the resource ID of the name set resource that is to be used as the pick exclusion filter during picking operations.

pick_data_rec

This attribute contains a pick-device-dependent data record used to initialize a pick measure resource when it is created.

prompt_echo_type

This attribute contains an enumerated type value that specifies the prompt/echo type to be used during picking operations. The allowable types are described in the "Extension Information" section.

echo_volume

This attribute 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 the PHIGS workstation resource at the time it was created.

echo_switch

This attribute specifies the initial echo state for a pick measure.

The requests in this section allow clients to set and inquire the values of pick device descriptors that are stored in a PHIGS workstation resource.

12.1.1. Get Pick Device Descriptor

Name:

PEXGetPickDevice

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
dev_type : PICK_DEVICE_TYPE
item_mask : BITMASK

Reply:

item_list : LISTofVALUE

Errors:

PhigsWKS, Value, FloatingPointFormat

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*.

12.1.2. Change Pick Device Descriptor

Name:

PEXChangePickDevice

Request:

fp_format : FLOAT_FORMAT
wks_id : PHIGS_WKS_ID
dev_type : PICK_DEVICE_TYPE
item_mask : BITMASK
item_list : LISTofVALUE

Errors:

PhigsWKS, Value, FloatingPointFormat, Path, NameSet

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*.

12.2. Pick Measure |

A *pick measure* resource must be created to actually perform a pick operation. A pick device type 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.

Attribute Name	**Data Type**	**Default Value**
pick_status	{*NoPick, Ok*}	from wks
picked_prim	LISTofPICK_ELEMENT_REF	from wks

The attributes of the pick measure resource are defined as follows:

pick_status
: This attribute 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.

picked_prim
: This attribute contains the path of the structure element that was picked as a result of the last update operation.

The pick measure is an X11 resource and carries all of the responsibilities and access rights of X11 resources. The requests in this section manage the creation and freeing of pick measures.

12.2.1. Create Pick Measure

Name:

PEXCreatePickMeasure

Request:

pm_id : PICK_MEASURE_ID
wks_id : PHIGS_WKS_ID
dev_type : PICK_DEVICE_TYPE

Errors:

IDChoice, PhigsWKS, Alloc, Value

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*.

12.2.2. Free Pick Measure *

Name:

PEXFreePickMeasure

Request:

pm_id : PICK_MEASURE_ID

Errors:

PickMeasure

Description:

This request deletes the pick measure resource and frees the storage associated with it.

12.2.3. Get Pick Measure Attributes *

Name:

PEXGetPickMeasure

Request:

pm_id : PICK_MEASURE_ID
item_mask : BITMASK

Reply:

item_list : LISTofVALUE

Errors:

PickMeasure, Value

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*.

12.2.4. Update Pick Measure *

Name:

PEXUpdatePickMeasure

Request:

pm_id : PICK_MEASURE_ID
input_record : LISTofCARD8

Errors:

PickMeasure, Path

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. *

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]

13. PEX Fonts |

The PEX font manipulation mechanisms are the same mechanisms as those used for X11 fonts. The PEX font resource is very similar to the font resource created by (and managed by) X11. PEX fonts that are used with PEX stroke precision text contain more functionality than is currently found in X11 fonts. Specifically, stroke precision text requires scalable and rotatable text fonts. Because of the added capabilities of PEX fonts, PEX has defined its own PEX font open, close, and query requests. The X11 font manipulation requests and the corresponding PEX font manipulation requests are listed below:

OpenFont augmented by PEXOpenFont
CloseFont augmented by PEXCloseFont
ListFonts augmented by PEXListFonts
ListFontsWithInfo augmented by PEXListFontsWithInfo
QueryFont augmented by PEXQueryFont
QueryTextExtents augmented by PEXQueryTextExtents

X11 fonts and PEX fonts can be stored in the same location on the server. PEX uses the X11 font path in order to find where PEX font files are located. (The X11 request **SetFontPath** is used to set the current font path and the X11 request **GetFontPath** is used to query the current font path.)

13.1. PEX Font Resource Management

The PEX font is an X11 resource and carries all of the responsibilities and access rights of X11 resources. These requests manage the opening and closing of PEX font resources.

13.1.1. Open PEX Font

Name:

PEXOpenFont

Request:

f_id : PEX_FONT_ID
name : STRING

Errors:

PEXFont, IDChoice, Alloc

Description:

This request loads the specified PEX font, if necessary, and associates identifier *f_id* with it. The font name should use the ISO Latin-1 encoding, and upper/lower case does not matter. PEX fonts are not associated with a particular screen, and can be used with any renderer or PHIGS workstation resources. An error will be generated if the specified font is not "PEX usable", that is, it is not capable of supporting the full range of PEX text attributes.

13.1.2. Close PEX Font

Name:

PEXCloseFont

Request:

f_id : PEX_FONT_ID

Errors:

PEXFont

Description:

This request deletes the association between the resource ID and the PEX font. The PEX font itself will be freed when no other resource references it.

13.2. PEX Font Inquiry

The PEX font requests generate replies with logical information specific to a font. The information is encoded in the following data structures.

PEX_FONTPROP is defined as:

name	: ATOM
value	: CARD32

PEX_FONTINFO is defined as:

first_glyph	: CARD32
last_glyph	: CARD32
default_glyph	: CARD32
all_glyphs_exist	: BOOLEAN
stroke_font	: BOOLEAN
properties	: LISTofPEX_FONTPROP

The *first_glyph*, *last_glyph*, and *default_glyph* are indices to the first glyph, last glyph, and default glyph of the font. The *default_glyph* specifies the glyph that will be displayed when an undefined or non-existent glyph is used. *Default_glyph* may specify a nonexistent glyph. In this case, if a client tries to display an undefined or non-existent glyph, nothing is drawn by the server (rather than having a default glyph drawn). If *all_glyphs_exist* is *True*, then all glyphs within the range of *first_glyph* and *last_glyph* have non-zero extents. *Stroke_font* is a flag indicating if the font is a PEX font and is provided so that the client can build font groups that all have the same text precision.

A font is not guaranteed to have any *properties*. Whether a property value is a signed or unsigned, and what units it is in must be derived from a priori knowledge of the property. It is strongly recommended that all fonts have at least a CHARSET_REGISTRY property (e.g., "ISO8859") and a CHARSET_ENCODING property (e.g., "1").

13.2.1. Query PEX Font

Name:

PEXQueryFont

Request:

f_id : PEX_FONT_ID

Reply:

font_info : PEX_FONTINFO

Errors:

PEXFont

Description:

This request generates a reply which contains the logical information about a PEX font.

13.2.2. List PEX Fonts

Name:

PEXListFonts

Request:

pattern : STRING
max_names : CARD16

Reply:

font_names : LISTofSTRING

Errors:

None

Description:

Like X11 **ListFonts** except that this request only returns the names of fonts that can support the full range of PEX text attributes (i.e., those fonts that are "PEX usable"). This list may or may not contain some of the same fonts returned by the X11 **ListFonts** request. This request returns a list of at most *max_names* entries, each of which contains information about a font matching the *pattern*. *Pattern* is a string that uses the ISO Latin-1 encoding, and upper/lower case does not matter. In the pattern, the '?' character (octal value 77) will match any single character, and the character '*' (octal value 52) will match any number of characters. The returned names are in lower case, and are also ISO Latin-1 encoded strings.

13.2.3. List PEX Fonts with Info

Name:

PEXListFontsWithInfo

Request:

pattern : STRING
max_names : CARD16

Reply:

font_names : LISTofSTRING
fonts : LISTofPEX_FONTINFO

Errors:

None

Description:

Like X11 **ListFontsWithInfo** except that this request only returns information about fonts that can support the full range of PEX text attributes (i.e., those fonts that are "PEX usable"). This list may or may not contain some of the same fonts returned by the X11 **ListFonts** request. This request returns a list of at most *max_names* entries, each of which contains information about a font matching the *pattern*. *Pattern* is a string that uses the ISO Latin-1 encoding, and upper/lower case does not matter. In the pattern, the '?' character (octal value 77) will match any single character, and the character '*' (octal value 52) will match any number of characters. The returned names are in lower case, and are also ISO Latin-1 encoded strings.

The information returned for each font is identical to what **PEXQueryFont** would return.

13.2.4. Query PEX Text Extents

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 : LISTofISTRING

Reply:

extents : LISTofEXTENT_INFO

Errors:

FloatingPointFormat, Value, Match

Description:

This request generates a reply which contains extent information in the local 2D text coordinate system for each of the specified strings. The extents of each string in *strings* is computed independently of the other strings in the list. If *resource_id* is a renderer or PHIGS workstation resource, the *TextFont* table used to perform the extents computation will be the *TextFont* table associated with the renderer or PHIGS workstation. If *resource_id* is a lookup table resource of type *TextFont*, it is used directly. *Font_group* provides the index of the entry that is to be used to obtain the font group. |

Stroke precision is assumed. The text position is (0,0) in the local 2D text coordinate system. *Concat_point* returns the point where the next glyph should go if the string is to be extended. This position is given in the 2D text coordinate system. (A suitable modeling transformation to account for the character up vector will still need to be applied by the client.)

If a specified font has no defined *default_glyph* (that is, if *default_glyph* refers to a non-existent glyph), then undefined glyphs in *strings* are taken to have all zero metrics.

Appendix A: Definition of Standard PEX Subsets

PEXGetExtensionInfo returns a 32-bit value (*subset_info*) containing information about whether the PEX server is a full PEX implementation or whether it supports some combination of the standard subsets. The top 16 bits of this 32-bit value are reserved for use by vendors. The bottom 16 bits contain information about how fully the PEX extension implementation supports the PEX protocol. Three PEX subsets are currently defined: "immediate rendering", "PHIGS workstation" and "structure rendering".

If all of the 16 low-order bits of *subset_info* are zero, then the extension is a complete PEX implementation, otherwise the bits are interpreted as follows:

bit 0 set - server supports all requests for "immediate rendering"
bit 1 set - server supports all requests for "PHIGS workstation"
bit 2 set - server supports all requests for "structure rendering"

For backward compatibility with PEX 5.0, a PEX implementation is not allowed to return with both bit 0 and bit 1 set.

If a server is sent a request that is *not* part of the subset(s) it supports, it will return a *Request* error.

Immediate Rendering Subset

An "immediate rendering" subset should fully support the following PEX resources:

- lookup tables
- name sets
- pipeline contexts
- renderers
- PEX fonts

and all of the output commands (the "execute structure" output command will be treated as a no-op). To qualify as an immediate rendering PEX subset, an implementation must support at least the protocol requests in the following list. Protocol requests that are not supported should return an *Implementation* error.

PEXGetExtensionInfo	**PEXCreateRenderer**
PEXGetEnumeratedTypeInfo	**PEXFreeRenderer**
PEXGetImpDepConstants	**PEXChangeRenderer**
PEXMatchRendererTargets	**PEXGetRendererAttributes**
	PEXGetRendererDynamics
	PEXBeginRendering
PEXCreateLookupTable	**PEXEndRendering**
PEXCopyLookupTable	**PEXBeginStructure**
PEXFreeLookupTable	**PEXEndStructure**
PEXGetTableInfo	**PEXRenderOutputCommands**
PEXGetPredefinedEntries	
PEXGetDefinedIndices	**PEXCreateNameSet**
PEXGetTableEntry	**PEXCopyNameSet**
PEXGetTableEntries	**PEXFreeNameSet**
PEXSetTableEntries	**PEXGetNameSet**
PEXDeleteTableEntries	**PEXChangeNameSet**
PEXCreatePipelineContext	**PEXQueryFont**
PEXCopyPipelineContext	**PEXListFontsWithInfo**
PEXFreePipelineContext	**PEXQueryTextExtents**

PEXGetPipelineContext |
PEXChangePipelineContext |

PEXBeginPickOne |
PEXEndPickOne |
PEXBeginPickAll |
PEXEndPickAll |

PEXListFonts |
PEXOpenFont |
PEXCloseFont |

PHIGS Workstation Subset

A "PHIGS workstation" subset should fully support the following PEX resources:

- lookup tables
- name sets
- structures
- search contexts
- PHIGS workstations
- pick measures
- PEX fonts

and all of the output commands. To qualify as a PHIGS workstation PEX subset, an implementation must support at least the protocol requests in the following list. Protocol requests that are not supported should return an *Implementation* error.

PEXGetExtensionInfo
PEXGetEnumeratedTypeInfo
PEXGetImpDepConstants
PEXMatchRendererTargets

PEXCreateNameSet
PEXCopyNameSet
PEXFreeNameSet
PEXGetNameSet
PEXChangeNameSet

PEXCreateLookupTable
PEXCopyLookupTable
PEXFreeLookupTable
PEXGetTableInfo
PEXGetPredefinedEntries
PEXGetDefinedIndices
PEXGetTableEntry
PEXGetTableEntries
PEXSetTableEntries
PEXDeleteTableEntries

PEXCreateSearchContext
PEXCopySearchContext
PEXFreeSearchContext
PEXGetSearchContext
PEXChangeSearchContext
PEXSearchNetwork

PEXCreateStructure
PEXCopyStructure
PEXDestroyStructures
PEXGetStructureInfo
PEXGetElementInfo
PEXGetStructuresInNetwork
PEXGetAncestors
PEXGetDescendants
PEXFetchElements
PEXSetEditingMode
PEXSetElementPointer
PEXSetElementPointerAtLabel
PEXElementSearch
PEXStoreElements
PEXDeleteElements
PEXDeleteElementsToLabel
PEXDeleteElementsBetweenLabels
PEXCopyElements
PEXChangeStructureReferences

PEXCreatePhigsWKS
PEXFreePhigsWKS
PEXGetWKSInfo
PEXGetDynamics
PEXGetViewRep
PEXRedrawAllStructures
PEXUpdateWorkstation
PEXRedrawClipRegion
PEXExecuteDeferredActions
PEXSetViewPriority
PEXSetDisplayUpdateMode
PEXMapDCtoWC
PEXMapWCtoDC
PEXSetViewRep
PEXSetWKSWindow
PEXSetWKSViewport
PEXSetHLHSRMode
PEXSetWKSBufferMode
PEXPostStructure
PEXUnpostStructure
PEXUnpostAllStructures
PEXGetWKSPostings

PEXGetPickDevice	**PEXQueryFont**
PEXChangePickDevice	**PEXListFontsWithInfo**
PEXCreatePickMeasure	**PEXQueryTextExtents**
PEXFreePickMeasure	**PEXListFonts**
PEXGetPickMeasure	**PEXOpenFont**
PEXUpdatePickMeasure	**PEXCloseFont**

Structure Rendering Subset

An "structure rendering" subset should fully support the following PEX resources:

- lookup tables
- name sets
- pipeline contexts
- renderers
- structures
- search contexts
- PEX fonts

and all of the output commands. To qualify as an structure rendering PEX subset, an implementation must support at least the protocol requests in the following list. Protocol requests that are not supported should return an *Implementation* error.

PEXGetExtensionInfo
PEXGetEnumeratedTypeInfo
PEXGetImpDepConstants
PEXMatchRendererTargets

PEXCreateLookupTable
PEXCopyLookupTable
PEXFreeLookupTable
PEXGetTableInfo
PEXGetPredefinedEntries
PEXGetDefinedIndices
PEXGetTableEntry
PEXGetTableEntries
PEXSetTableEntries
PEXDeleteTableEntries

PEXCreatePipelineContext
PEXCopyPipelineContext
PEXFreePipelineContext
PEXGetPipelineContext
PEXChangePipelineContext

PEXCreateRenderer
PEXFreeRenderer
PEXChangeRenderer
PEXGetRendererAttributes
PEXGetRendererDynamics
PEXBeginRendering
PEXEndRendering
PEXRenderNetwork
PEXRenderElements
PEXAccumulateState

PEXBeginPickOne
PEXEndPickOne
PEXPickOne
PEXBeginPickAll
PEXEndPickAll

PEXCreateStructure
PEXCopyStructure
PEXDestroyStructures
PEXGetStructureInfo
PEXGetElementInfo
PEXGetStructuresInNetwork
PEXGetAncestors
PEXGetDescendants
PEXFetchElements
PEXSetEditingMode
PEXSetElementPointer
PEXSetElementPointerAtLabel
PEXElementSearch
PEXStoreElements
PEXDeleteElements
PEXDeleteElementsToLabel
PEXDeleteElementsBetweenLabels
PEXCopyElements
PEXChangeStructureReferences

PEXCreateNameSet
PEXCopyNameSet
PEXFreeNameSet
PEXGetNameSet
PEXChangeNameSet

PEXCreateSearchContext
PEXCopySearchContext
PEXFreeSearchContext
PEXGetSearchContext
PEXChangeSearchContext
PEXSearchNetwork

PEXQueryFont
PEXListFontsWithInfo
PEXQueryTextExtents
PEXListFonts
PEXOpenFont
PEXCloseFont

Appendix B: Minimum Support for PHIGS/PHIGS-PLUS

In order to fully support a PHIGS client-side implementation and the targeted PHIGS-PLUS functionality, a PEX extension implementation must meet or exceed the following support criteria. Other than the minimum support criteria listed here and the subset information listed in Appendix A, a PEX extension implementation must fully support the functionality described in the *PEX Protocol Specification* document.

Enumerated Types

MarkerType	at least types 1-5 (*Dot, Cross, Asterisk, Circle, X*) must be supported
ATextStyle	at least types 1-2 (*NotConnected, Connected*) must be supported
InteriorStyle	at least types 1,2,5 (*Hollow, Solid, Empty*) must be supported
HatchStyle	it is not required that an implementation support any hatch styles
LineType	at least types 1-4 (*Solid, Dashed, Dotted, DashDot*) must be supported
SurfaceEdgeType	at least type 1 (*Solid*) must be supported
PickDeviceType	at least pick device type 1 (*DC_HitBox*) must be supported (for full implementation or for the "PHIGS workstation only" subset)
PolylineInterpMethod	at least types 1-2 (*None, Color*) must be supported
CurveApproxMethod	at least types 1-8 (*imp. dep., ConstantBetweenKnots, WCS_ChordalSize, NPC_ChordalSize, DC_ChordalSize, WCS_ChordalDev, NPC_ChordalDev, DC_ChordalDev*) must be supported
ReflectionModel	at least types 1-4 (*NoShading, Ambient, Diffuse, Specular*) must be supported
SurfaceInterpMethod	at least types 1-2 (*None, Color*) must be supported
SurfaceApproxMethod	at least types 1-8 (*imp. dep., ConstantBetweenKnots, WCS_ChordalSize, NPC_ChordalSize, DC_ChordalSize, WCS_PlanarDev, NPC_PlanarDev, DC_PlanarDev*) must be supported
TrimCurveApproxMethod	at least types 1-2 (*imp. dep., ConstantBetweenKnots*) must be supported
ModelClipOperator	at least types 1-2 (*Replace, Intersection*) must be supported
LightType	at least types 1-4 (*Ambient, WCS_Vector, WCS_Point, WCS_Spot*) must be supported
ColorType	type 0 (*Indexed*) and types 1 (*RGBFloat*) must be supported
FloatFormat	at least one of types 1-2 (*IEEE_754_32, DEC_F_Floating*) must be supported
HLHSRMode	at least type 1 (*Off*) and one other type (*ZBuffer, Painters, Scanline, HiddenLineOnly*, or an implementation-dependent type) must be supported
PromptEchoType	at least type 1 (*EchoPrimitive*) must be supported
DisplayUpdateMode	at least types 1-3 (*VisualizeEach, VisualizeEasy, VisualizeNone*) must be supported
ColorApproxType	at least one of types 1-2 {*ColorSpace, ColorRange*} must be supported
ColorApproxModel	at least one of types 1-5 {*RGB, HSV, HLS, CIE, YIQ*} must be supported
RenderingColorModel	at least type 0 (implementation-dependent) must be supported
ParametricSurfaceCharacteristics	at least types 1-3 (*None, imp. dep., IsoparametricCurves*) must be supported
BufferMode	it is not required that an implementation support double buffering
GDP	it is not required that an implementation support any GDPs
GDP3	it is not required that an implementation support any GDP3s
GSE	it is not required that an implementation support any GSEs

Output Primitive Attributes

marker_scale	marker scale factors other than 1.0 may be ignored
text_precision	all types (*Stroke, Char, String*) must be supported
char_expansion	character expansion other than 1.0 may be ignored
char_height	character height other than 0.01 may be ignored
atext_height	annotation text character height other than 0.01 may be ignored

line_width	line widths other than 1.0 may be ignored
interior_style_index	this attribute may be ignored if neither *Pattern* nor *Hatch* style is supported
reflection_attr	transmission coefficient attribute may be ignored
bf_interior_style_index	this attribute may be ignored if neither *Pattern* nor *Hatch* style is supported
bf_reflection_attr	transmission coefficient attribute may be ignored
pattern_size	this attribute may be ignored if *Pattern* style is not supported
pattern_ref_pt	this attribute may be ignored if *Pattern* style is not supported
pattern_ref_vec1	this attribute may be ignored if *Pattern* style is not supported
pattern_ref_vec2	this attribute may be ignored if *Pattern* style is not supported
surface_edge_width	surface edge widths other than 1.0 may be ignored
model_clip_volume	must support the combining of at least six halfspaces to compute the modeling clipping volume
depth_cue attributes	depth cue attributes other than the default (depth cue off) may be ignored
HLHSR_identifier	at least one must be available
specific GSEs	may be ignored

Output Primitives

NURB surfaces	must be supported up to order six
NURB curves	must be supported up to order six
trim curves	must be supported up to order four
cell arrays	may be simulated by drawing the outline of the cell array
specific GDPs	may be ignored

Lookup Tables

LineBundle	must support at least 5 entries
MarkerBundle	must support at least 5 entries
TextBundle	must support at least 6 entries
InteriorBundle	must support at least 5 entries
EdgeBundle	must support at least 5 entries
Pattern	must support at least 10 entries (if interior style *Pattern* supported)
Color	must support at least 2 entries
TextFont	must support at least 2 entries
View	must support at least 6 entries
Light	must support at least 2 entries
DepthCue	must support at least 2 entries
ColorApprox	must support at least 1 entry

Miscellaneous

fonts	an implementation need not support drawing text primitives with X11 fonts font *one* must be mono-spaced
namesets	at least 64 names must be supported
search contexts	at least one normal and one inverted filter list must be supported
ws display priorities	at least two ws display priorities must be supported

Appendix C: Definition of PEX Errors

PEXGetExtensionInfo
none

PEXGetEnumeratedTypeInfo
Drawable: specified drawable resource ID is invalid
Value: specified enumerated type number is invalid
Match: specified drawable is not supported |

PEXGetImpDepConstants
Value: a specified constant name is invalid
FloatingPointFormat: device does not support the specified fp format
Drawable: specified drawable resource ID is invalid
Match: specified drawable is not supported |

PEXMatchRendererTargets |
Drawable: specified drawable resource ID is invalid |
Value: specified visual is invalid |

PEXEscape |
Value: a specified constant name is invalid |

PEXEscapeWithReply |
Value: a specified constant name is invalid |

PEXCreateLookupTable
IDChoice: ID already in use or not in range assigned to client
Drawable: specified drawable resource ID is invalid
Match: specified drawable is not supported |
Value: table_type value does not name a valid table type
Alloc: server failed to allocate the requested resource
LookupTable: table type not supported by implementation

PEXCopyLookupTable
LookupTable: either src_lut_id or dest_lut_id is an invalid resource ID
LookupTable: table type not supported by implementation
Match: src_lut_id and dest_lut_id must have been created for use on the same class of drawables, and must be the same type of lookup table

PEXFreeLookupTable
LookupTable: lut_id contains an invalid lut resource ID

PEXGetTableInfo
Drawable: specified drawable resource ID is invalid
Match: specified drawable is not supported |
Value: table_type value does not name a valid table type
LookupTable: table type not supported by implementation |

PEXGetPredefinedEntries
Drawable: specified drawable resource ID is invalid

Match: specified drawable is not supported |
Value: table_type value does not name a valid table type
Value: start < min predefined entry
Value: start + count > max predefined entry
Value: entry 0 not valid for this table type
FloatingPointFormat: device does not support the specified fp format
LookupTable: table type not supported by implementation

PEXGetDefinedIndices

LookupTable: lut_id contains an invalid lut resource ID
LookupTable: table type not supported by implementation

PEXGetTableEntry

LookupTable: lut_id contains an invalid lut resource ID
LookupTable: table type not supported by implementation
FloatingPointFormat: device does not support the specified fp format
Value: entry 0 not valid for this table type

PEXGetTableEntries

LookupTable: lut_id contains an invalid lut resource ID
LookupTable: table type not supported by implementation
Value: start + count is greater than 65535
Value: entry 0 not valid for this table type
FloatingPointFormat: device does not support the specified fp format

PEXSetTableEntries

LookupTable: lut_id contains an invalid lut resource ID
LookupTable: table type not supported by implementation
Value: start + count is greater than 65535
Value: illegal value in one of the fields of a table entry
Value: entry 0 not valid for this table type
FloatingPointFormat: device does not support the specified fp format
ColorType: device does not support the specified color type
ColorType: color type may not be *Indexed* for color lookup table
Alloc: table is full

PEXDeleteTableEntries

LookupTable: lut_id contains an invalid lut resource ID
LookupTable: table type not supported by implementation
Value: start + count is greater than 65535
Value: entry 0 not valid for this table type

PEXCreatePipelineContext

IDChoice: ID already in use or not in range assigned to client
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format
ColorType: device does not support the specified color type
Alloc: server failed to allocate the requested resource
NameSet: a specified name set resource ID is invalid |

PEXCopyPipelineContext

PipelineContext: pc_id contains an invalid pipeline context ID

Value: illegal bit set in item mask parameter

PEXFreePipelineContext

PipelineContext: pc_id contains an invalid pipeline context ID

PEXGetPipelineContext

PipelineContext: pc_id contains an invalid pipeline context ID
FloatingPointFormat: device does not support the specified fp format
Value: illegal bit set in item mask parameter

PEXChangePipelineContext

PipelineContext: pc_id contains an invalid pipeline context ID
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format
ColorType: device does not support the specified color type
NameSet: a specified name set resource ID is invalid |

PEXCreateRenderer

IDChoice: ID already in use or not in range assigned to client
Drawable: specified drawable resource ID is invalid
PipelineContext: specified pipeline context resource ID is invalid
NameSet: a specified name set resource ID is invalid
LookupTable: a specified lookup table resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
Alloc: server failed to allocate the requested resource
Match: lookup table root/depth does not match example drawable's root/depth
Match: specified drawable is not supported |

PEXFreeRenderer

Renderer: specified renderer resource ID is invalid

PEXChangeRenderer

Renderer: specified renderer resource ID is invalid
Match: specified lookup table resource was not created for drawables
of the same root and depth as the specified renderer
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format
NameSet: a specified name set resource ID is invalid
LookupTable: a specified lookup table resource ID is invalid
PipelineContext: specified pipeline context resource ID is invalid

PEXGetRendererAttributes

Renderer: specified renderer resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: illegal bit set in item mask parameter

PEXGetRendererDynamics

Renderer: specified renderer resource ID is invalid

PEXBeginRendering

Renderer: specified renderer resource ID is invalid
Drawable: specified drawable resource ID is invalid
Match: specified renderer resource was not created for drawables of the same root and depth as the specified drawable |
Match: specified drawable is not supported |
Alloc: server was unable to allocate the resources necessary to do rendering
RendererState: renderer was in the *Rendering* or *Picking* state |

PEXEndRendering

Renderer: specified renderer resource ID is invalid
RendererState: renderer was in the *Picking* state |

PEXBeginStructure

Renderer: specified renderer resource ID is invalid

PEXEndStructure

Renderer: specified renderer resource ID is invalid
RendererState: no matching begin structure

PEXRenderOutputCommands

Renderer: specified renderer resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
OutputCommand: illegal value in output commands

PEXRenderElements |

Renderer: specified renderer resource ID is invalid |
Structure: specified structure resource ID is invalid |
Value: bad value for "whence" parameter |

PEXAccumulateState |

Renderer: specified renderer resource ID is invalid |
Path: illegal or poorly-formed search path (includes invalid structure IDs, invalid element offset values) |

PEXRenderNetwork

Renderer: specified renderer resource ID is invalid
Drawable: specified drawable resource ID is invalid
Structure: specified structure resource ID is invalid
Match: specified renderer resource was not created for drawables of the same root and depth as the specified drawable |
Match: specified drawable is not supported |
Alloc: server was unable to allocate the resources necessary to do rendering |
RendererState: renderer was in the *Rendering* or *Picking* state |

PEXBeginPickOne |

Renderer: specified renderer resource ID is invalid |
Drawable: specified drawable resource ID is invalid |
Match: specified renderer resource was not created for drawables of the same root and depth as the specified drawable |
Match: specified drawable is not supported |
Alloc: server was unable to allocate the resources necessary to do picking |
RendererState: renderer was in the *Rendering* or *Picking* state |

Value: bad value given for "pick_type" or "pick_data" |

PEXEndPickOne |
Renderer: specified renderer resource ID is invalid |
RendererState: renderer was in the *Rendering* state |

PEXPickOne |
Renderer: specified renderer resource ID is invalid |
Drawable: specified drawable resource ID is invalid |
Structure: specified structure resource ID is invalid |
Match: specified renderer resource was not created for drawables |
of the same root and depth as the specified drawable |
Match: specified drawable is not supported |
Alloc: server was unable to allocate the resources necessary to do picking |
RendererState: renderer was in the *Rendering* or *Picking* state |
Value: bad value given for "pick_type" or "pick_data" |

PEXBeginPickAll |
Renderer: specified renderer resource ID is invalid |
Drawable: specified drawable resource ID is invalid |
Match: specified renderer resource was not created for drawables |
of the same root and depth as the specified drawable |
Match: specified drawable is not supported |
Alloc: server was unable to allocate the resources necessary to do picking |
RendererState: renderer was in the *Rendering* or *Picking* state |
Value: bad value given for "pick_data" |

PEXEndPickAll |
Renderer: specified renderer resource ID is invalid |
RendererState: renderer was in the *Rendering* state |

PEXPickAll |
Renderer: specified renderer resource ID is invalid |
Drawable: specified drawable resource ID is invalid |
Structure: specified structure resource ID is invalid |
Match: specified renderer resource was not created for drawables |
of the same root and depth as the specified drawable |
Match: specified drawable is not supported |
Alloc: server was unable to allocate the resources necessary to do picking |
RendererState: renderer was in the *Rendering* or *Picking* state |
Value: bad value given for "pick_data" |
Path: initial pick path is invalid |

PEXCreateStructure
IDChoice: ID already in use or not in range assigned to client
Alloc: server failed to allocate the requested resource

PEXCopyStructure
Structure: specified structure resource ID is invalid

PEXDestroyStructures
Structure: specified structure resource ID is invalid

PEXGetStructureInfo

Structure: specified structure resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: illegal bit set in item_mask parameter

PEXGetElementInfo

Structure: specified structure resource ID is invalid
Value: bad value for "whence" parameter
FloatingPointFormat: device does not support the specified fp format

PEXGetStructuresInNetwork

Structure: specified structure resource ID is invalid
Value: bad value for "which" parameter

PEXGetAncestors

Structure: specified structure resource ID is invalid
Value: bad value for "path_part" parameter

PEXGetDescendants

Structure: specified structure resource ID is invalid
Value: bad value for "path_part" parameter

PEXFetchElements

Structure: specified structure resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: bad value for "whence" parameter

PEXSetEditingMode

Structure: specified structure resource ID is invalid
Value: bad value for "mode" parameter

PEXSetElementPointer

Structure: specified structure resource ID is invalid
Value: bad value for "whence" parameter

PEXSetElementPointerAtLabel

Structure: specified structure resource ID is invalid
Label: no occurrences of specified label in structure

PEXElementSearch

Structure: specified structure resource ID is invalid
Value: bad value for "whence" or "direction" parameters

PEXStoreElements

Structure: specified structure resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
OutputCommand: illegal value in output commands

PEXDeleteElements

Structure: specified structure resource ID is invalid
Value: bad value for "whence" parameter

PEXDeleteElementsToLabel

Structure: specified structure resource ID is invalid
Label: no occurrences of specified label in structure
Value: bad value for "whence" parameter

PEXDeleteElementsBetweenLabels
Structure: specified structure resource ID is invalid
Label: no occurrences of specified label in structure

PEXCopyElements
Structure: specified structure resource ID is invalid
Value: bad value for "whence" parameter

PEXChangeStructureReferences
Structure: specified structure resource ID is invalid

PEXCreateNameSet
IDChoice: ID already in use or not in range assigned to client
Alloc: server failed to allocate the requested resource

PEXCopyNameSet
NameSet: specified name set resource ID is invalid

PEXFreeNameSet
NameSet: specified name set resource ID is invalid

PEXGetNameSet
NameSet: specified name set resource ID is invalid

PEXChangeNameSet
NameSet: specified name set resource ID is invalid
Value: bad value for "action" parameter

PEXCreateSearchContext
IDChoice: ID already in use or not in range assigned to client
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format
Alloc: server failed to allocate the requested resource
Path: illegal or poorly-formed search path (includes invalid structure IDs, invalid element offset values)
NameSet: specified name set resource ID is invalid

PEXCopySearchContext
SearchContext: specified search context resource ID is invalid
Value: illegal bit set in item mask parameter

PEXFreeSearchContext
SearchContext: specified search context resource ID is invalid

PEXGetSearchContext
SearchContext: specified search context resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: illegal bit set in item mask parameter

PEXChangeSearchContext
SearchContext: specified search context resource ID is invalid
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format
Path: illegal or poorly-formed search path (includes invalid structure IDs, invalid element offset values)
NameSet: specified name set resource ID is invalid

PEXSearchNetwork
SearchContext: specified search context resource ID is invalid
Path: illegal or poorly-formed search path (includes invalid structure IDs, invalid element offset values)

PEXCreatePhigsWKS
IDChoice: ID already in use or not in range assigned to client
Drawable: specified drawable resource ID is invalid
Match: specified lookup table resource was not created for drawables of the same root and depth as the specified drawable
Match: specified drawable is not supported |
LookupTable: a specified lookup table resource ID is invalid
NameSet: a specified name set resource ID is invalid
Alloc: server failed to allocate the requested resource
Alloc: server cannot allocate resources necessary for double-buffering
Value: bad value for buffer_mode parameter

PEXFreePhigsWKS
PhigsWKS: specified PHIGS workstation resource ID is invalid

PEXGetWKSInfo
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: illegal bit set in item mask parameter

PEXGetDynamics
Drawable: specified drawable resource ID is invalid
Match: specified drawable is not supported |

PEXGetViewRep
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: specified view table entry is not defined

PEXRedrawAllStructures
PhigsWKS: specified PHIGS workstation resource ID is invalid

PEXUpdateWorkstation
PhigsWKS: specified PHIGS workstation resource ID is invalid

PEXRedrawClipRegion
PhigsWKS: specified PHIGS workstation resource ID is invalid

PEXExecuteDeferredActions

PhigsWKS: specified PHIGS workstation resource ID is invalid

PEXSetViewPriority
PhigsWKS: specified PHIGS workstation resource ID is invalid
Value: bad value for "priority" parameter
Value: specified table entry is not defined

PEXSetDisplayUpdateMode
PhigsWKS: specified PHIGS workstation resource ID is invalid
Value: bad value for "display_update" parameter

PEXMapWCtoDC
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format

PEXMapDCtoWC
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format

PEXSetViewRep
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Alloc: table is full

PEXSetWKSWindow
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format

PEXSetWKSViewport
PhigsWKS: specified PHIGS workstation resource ID is invalid
FloatingPointFormat: device does not support the specified fp format
Value: bad value for "use_drawable" parameter

PEXSetHLHSRMode
PhigsWKS: specified PHIGS workstation resource ID is invalid
Value: bad value for "mode" parameter
Alloc: server cannot allocate necessary resources |

PEXSetWKSBufferMode
PhigsWKS: specified PHIGS workstation resource ID is invalid
Value: bad or unsupported value for "buffer_mode" parameter |
Alloc: server cannot allocate resources necessary for double-buffering

PEXPostStructure
PhigsWKS: specified PHIGS workstation resource ID is invalid
Structure: specified structure resource ID is invalid
FloatingPointFormat: device does not support the specified fp format

PEXUnpostStructure
PhigsWKS: specified PHIGS workstation resource ID is invalid
Structure: specified structure resource ID is invalid

PEXUnpostAllStructures

PhigsWKS: specified PHIGS workstation resource ID is invalid

PEXGetWKSPostings

Structure: specified structure resource ID is invalid

PEXGetPickDevice

PhigsWKS: specified PHIGS workstation resource ID is invalid
Value: bad value for "dev_type" parameter
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format

PEXChangePickDevice

PhigsWKS: specified PHIGS workstation resource ID is invalid
Value: bad value for "dev_type" parameter
Value: an item in the item_list is out of range
Value: illegal bit set in item mask parameter
FloatingPointFormat: device does not support the specified fp format
Path: illegal or poorly-formed pick path (includes invalid structure IDs, invalid element offset values)
NameSet: a specified name set resource ID is invalid

PEXCreatePickMeasure

IDChoice: ID already in use or not in range assigned to client
PhigsWKS: specified PHIGS workstation resource ID is invalid
Alloc: server failed to allocate the requested resource
Value: bad value for "dev_type" parameter

PEXFreePickMeasure

PickMeasure: specified pick measure resource ID is invalid

PEXGetPickMeasure

PickMeasure: specified pick measure resource ID is invalid
Value: illegal bit set in item mask parameter

PEXUpdatePickMeasure

PickMeasure: specified pick measure resource ID is invalid
Path: illegal or poorly-formed search path (includes invalid structure IDs, invalid element offset values)

PEXOpenFont

PEXFont: string does not name a useable PEX font
IDChoice: ID already in use or not in range assigned to client
Alloc: server failed to allocate the requested resource

PEXCloseFont

PEXFont: specified PEX font resource ID is invalid

PEXQueryFont

PEXFont: specified PEX font resource ID is invalid

PEXListFonts

none

PEXListFontsWithInfo
none

PEXQueryTextExtents
FloatingPointFormat: device does not support the specified fp format
Value: input text attribute values are illegal or out of range
Value: resource ID does not name a valid renderer, PHIGS workstation
or lookup table
Match: resource ID specifies a table of type other than TextFont

Appendix D: Definition of Table Default Values

If a table entry that is not defined is referenced, the contents of the default table entry will be used. The default entry for all tables is one, except for the view, depth cue, and color approximation tables whose default entry is zero. If the contents of the default table entry is not defined, then the default attribute values listed below will be used to define a default table entry that will be used instead. For each type of table, the attribute name, data type, and default value are listed. Each PEX implementation should provide documentation describing the choices it made for those attributes listed as "implementation-dependent".

LineBundle (1..65535, default entry = 1)

line_type	LINE_TYPE	*LineTypeSolid*
polyline_interp	POLYLINE_INTERP	*PolylineInterpNone*
curve_approx	CURVE_APPROX	{1, 1.0}†
line_width	FLOAT	1.0
line_color	COLOR_SPECIFIER	{*Indexed*, 1}

MarkerBundle (1..65535, default entry = 1)

marker_type	MARKER_TYPE	*MarkerAsterisk*
marker_scale	FLOAT	1.0
marker_color	COLOR_SPECIFIER	{*Indexed*, 1}

TextBundle (1..65535, default entry = 1)

text_font_index	TABLE_INDEX	1
text_precision	TEXT_PRECISION	*String*
char_expansion	FLOAT	1.0
char_spacing	FLOAT	0.0
text_color	COLOR_SPECIFIER	{*Indexed*, 1}

InteriorBundle (1..65535, default entry = 1)

interior_style	INTERIOR_STYLE	*InteriorStyleHollow*
interior_style_index	TYPE_OR_TABLE_INDEX	1
surface_color	COLOR_SPECIFIER	{*Indexed*, 1}
reflection_attr	REFLECTION_ATTR	{1.0, 1.0, 1.0, 0.0, 0.0, (*Indexed*, 1)}
reflection_model	REFLECTION_MODEL	*ReflectionNoShading*
surface_interp	SURFACE_INTERP	*SurfaceInterpNone*
bf_interior_style	INTERIOR_STYLE	*InteriorStyleHollow*
bf_interior_style_index	TYPE_OR_TABLE_INDEX	1
bf_surface_color	COLOR_SPECIFIER	{*Indexed*, 1}
bf_reflection_attr	REFLECTION_ATTR	{1.0, 1.0, 1.0, 0.0, 0.0, (*Indexed*, 1)}
bf_reflection_model	REFLECTION_MODEL	*ReflectionNoShading*
bf_surface_interp	SURFACE_INTERP	*SurfaceInterpNone*
surface_approx	SURFACE_APPROX	{1, 1.0, 1.0}

† PHIGS-PLUS defines the default curve approximation type to be 1, which is an implementation-dependent method. |

EdgeBundle (1..65535, default entry = 1)

surface_edges	SWITCH	*Off*
surface_edge_type	SURFACE_EDGE_TYPE	*SurfaceEdgeSolid*
surface_edge_width	FLOAT	1.0
surface_edge_color	COLOR_SPECIFIER	{*Indexed*, 1}

Pattern (1..65535, default entry = 1)

color_type	COLOR_TYPE	implementation-dependent
numx	CARD16	implementation-dependent
numy	CARD16	implementation-dependent
colors	LISTofCOLOR	implementation-dependent

Color (0..65534, default entry = 1)

color_type	COLOR_TYPE	implementation-dependent
color	DIRECT_COLOR	implementation-dependent

TextFont (1..65535, default entry = 1)

font	LISTofFONT_ID	implementation-dependent

View (0..65534, default entry = 0)

clip_flags	BITMASK_SHORT	all *On*
clip_limits	NPC_SUBVOLUME	(0,0,0),(1,1,1)
orientation	MATRIX	identity matrix
mapping	MATRIX	identity matrix

Light (1..65535, default entry = 1)

light_type	LIGHT_TYPE	implementation-dependent
direction	VECTOR_3D	implementation-dependent
point	COORD_3D	implementation-dependent
concentration	FLOAT	implementation-dependent
spread_angle	FLOAT	implementation-dependent
attenuation	[factor1, factor2 : FLOAT]	implementation-dependent
color	COLOR_SPECIFIER	implementation-dependent

Depending on the type of light, some of the values in a table entry may be ignored. Undefined entries in the light table that are referenced are treated as lights that are set to *Off*.

DepthCue (0..65534, default entry = 0)

mode	SWITCH	*Off*
front_plane	FLOAT	implementation-dependent
back_plane	FLOAT	implementation-dependent
front_scaling	FLOAT	implementation-dependent
back_scaling	FLOAT	implementation-dependent
color	COLOR_SPECIFIER	implementation-dependent

ColorApprox (0..65534, default entry = 0)

type	COLOR_APPROX_TYPE	implementation-dependent
color_model	COLOR_APPROX_MODEL	implementation-dependent
max1	CARD16	implementation-dependent
max2	CARD16	implementation-dependent
max3	CARD16	implementation-dependent
mult1	CARD32	implementation-dependent
mult2	CARD32	implementation-dependent
mult3	CARD32	implementation-dependent
weight1	FLOAT	implementation-dependent
weight2	FLOAT	implementation-dependent
weight3	FLOAT	implementation-dependent
base_pixel	CARD32	implementation-dependent
dither	SWITCH	implementation-dependent

It is suggested that PEX implementations provide default entries that correlate with the X definition of RGB_DEFAULT_MAP for the device. For instance, if a device has eight-bit pixels which index into a 24-bit hardware colormap, and its notion of an RGB_DEFAULT_MAP is a 6×6×6 color cube encoded in 216 colormap entries starting with cell number 16, then the suggested values might be:

type	COLOR_APPROX_TYPE	*ColorSpace*
color_model	COLOR_APPROX_MODEL	*RGB*
max1	CARD16	5
max2	CARD16	5
max3	CARD16	5
mult1	CARD32	1
mult2	CARD32	6
mult3	CARD32	36
weight1	FLOAT	1.0
weight2	FLOAT	1.0
weight3	FLOAT	1.0
base_pixel	CARD32	16
dither	SWITCH	*Off*

Appendix E: Registered PEX Escapes

A PEX implementation is not required to support any of the registered PEX escapes. The list of supported escapes which a particular PEX implementation supports can be inquired by calling **PEXGetEnumeratedTypeInfo**. All registered escapes contain an *escape_id*-- this is a positive number which identifies the escape request.

Set Echo Color

Name:

PEXSetEchoColor

Request:

escape_id : CARD32
fp_format : FLOAT_FORMAT
rdr_id : RENDERER_ID
echo_color : COLOR_SPECIFIER

Errors:

Renderer, ColorType, FloatingPointFormat

Description:

This request changes the renderer's *echo_color* attribute. The modification is dynamic and takes effect immediately. The echo color is used to renderer primitives when the renderer's *echo_mode* is set to *Echo*. The lighting and shading for echoed primitives is implementation-dependent.

echo_color is a new renderer attribute introduced by this request and the concept of an echo color may not be supported by PEX implementations which do not support this escape. The *echo_color* renderer attribute cannot be inquired and the default value is implementation-dependent.

If the specified renderer identifier is invalid then a *Renderer* error is generated. A *ColorType* error is returned if the specified color type (in the *echo_color*) is not supported and a *FloatingPointFormat* error is returned if the specified floating point format is not supported. The supported color types and floating point formats can be inquired by calling **PEXGetEnumeratedTypeInfo**.