Solaris PEX Implementation Specification

beta draft

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Contents

Figures

Tables

Introduction

1

The Solaris® PEX™ package contains the Sun Microsystems implementation of PEXlib and PEX Server extension; PEXlib is an industry standard 3D graphics application program interface (API) library for accessing PEX; PEX is a protocol extension to the X Windows protocol for supporting distributed 3D graphics.

The current release of Solaris PEX PEXlib has PEX 5.1 support, and it has a performance enhancement for applications running locally.

The current release of Solaris PEX server has PEX 5.1 support, and it is a dynamically loadable, shared-object module that is automatically loaded when the first PEX protocol request is received by the OpenWindows X11 server.

The Solaris PEX server is a *display server*—that is, it generates and controls the display on which your application draws 3D graphics. The application program that specifies the drawing may or may not be running on the same machine as the server. If the application and server are running on the same machine, your application is running *locally* (that is, "non-networked"). If the application and server are running on different machines, your application is running *remotely* (that is, over a network).

It is important to recognize the context in which the term *server* is used in this document. When we refer to the Solaris PEX server, we mean the display server, *not* a traditional application server or file server. In fact, we typically refer to the application that sends PEX requests to a Solaris PEX display server as a *client*. This is consistent with the X Window System™ usage of the terms *server* and *client*.

PEXlib and the server extension:

- Operate in the Solaris environment
- Provide networked access to 3D graphics in the OpenWindows X11 server
- Provide a compliant PEX 5.1 implementation
- Provide support for Common Open Software Environment PEX 5.1 extensions
- Provide the Evans & Sutherland and Hewlett-Packard double-buffering escape extensions
- Provide additional Sun extensions through PEX Escapes, GDPs and GSEs
- Provide support for Independent Hardware Vendors's (IHV's) frame buffers via the Sun Open Graphics Initiative (OGI)

Note – The PHIGS Workstation subset is not supported in this release.

Environment Variables Used by Solaris PEX

SUNPEX_DIRECT_ENABLE

This environment variable is applicable when running PEXlib applications locally, that is, on the same machine that the PEX server is running. Setting this environment variable enables PEXlib to use a direct path to the graphics capabilities on your machine without having to pass through the PEX protocol interface to the server. The PEX protocol interface requires that PEXlib encode your application data, and that the PEX server receiving it decode it. Hence, setting this environment variable before running your application (locally) eliminates the overhead associated with encoding and decoding data, and may result in improved performance. In addition, several performance enhancements have been added to the direct path, therefore; in certain circumstances, it is possible to realize a substantial improvement in performance by setting this environment variable.

SUNPEX_USE_VERTEX_COLOR

When *PEXSetSurfaceInterpMethod* (or *PEXSetBFSurfaceInterpMethod* for back-facing facets) is set to *PEXSurfaceInterpNone*, i.e. when shading is turned off, the PEX specification dictates that graphics primitives be rendered using the color of the first vertex, the facet color (if vertex color data is not present), or the context's surface color (if neither vertex nor facet color data is present). However, on GX and GX+ type framebuffers, when shading is turned off, primitives are rendered using the context's surface color, while the vertex and facet colors of the primitive are completely ignored. The user can force PEX to select the primitive color in accordance with the PEX specification by setting this environment variable. Note that there is a performance trade-off involved here, and that setting this environment variable may adversely affect performance.

SUNPEX_XGL_ERROR_DETECT

This environment variable enables or disables additional PEX server internal error checking. When this environment variable is set, additional error checking tests are executed as part of any PEX call and additional diagnostic error messages may be generated. Otherwise, the additional error checking tests are not performed. Note that there is a performance trade-off involved here, and that setting this environment variable may adversely affect performance.

SUNPEX_MAX_STRUCT_DEPTH

This environment variable can be used to set the maximum levels of structure hierarchy that PEX can traverse. The default value is 1024.

Colormap Flashing

In order to avoid colormap flashing, it is recommended that a standard colormap defined in the standard colormap properties be used. The CGE color utilities included in the CGE example directory assumes that the `RGB_DEFAULT_MAP` property contains an entry for all the PEX supported visuals. Solaris PEX only supports PseudoColor visuals on 8-bit devices. To ensure that there is an entry for the PseudoColor visual in the `RGB_DEFAULT_MAP` property, run *xstdcmap -default* from the X startup script.

PHIGS Workstation Subset (PWS) Deprecation

This version of Solaris PEX does not support the PHIGS Workstation subset of PEX. Users are encouraged to port their existing PHIGS workstation subset applications to the immediate mode subset or the structure rendering subset.

Execution Requirements

To run Solaris PEX, your system needs the following:

- The most current releases of Solaris operating environment and OpenWindows.
- The most current release of XGL

To install Solaris PEX, refer to the *Software Developer Kit Installation Guide*.

PEXlib 2

PEXlib is an application program interface (API) to the PEX protocol. It is a programmer's interface to the PEX protocol just as Xlib is to the X protocol. It provides application portability across hardware platforms and enables 3D graphics rendering on local and remote displays. The PEXlib implementation for Solaris PEX is based on the *PEXlib Specification and C Language Binding Version 5.1* and *CGE PEX 5.1 Specification*.

Direct PEXlib

Direct PEXlib is a performance enhancement mode of Sun's PEXlib implementation that you can use when you display PEXlib applications locally, that is, on the same machine as your PEX client. To enable Direct PEXlib, set the environment variable `SUNPEX_DIRECT_ENABLE` in your environment before starting a PEXlib client:

```
% setenv SUNPEX_DIRECT_ENABLE
```

Setting this environment variable enables PEXlib to use a direct path to graphics capabilities in your machine, without passing through the PEX protocol interface to the server. The PEX protocol interface requires that PEXlib encode your application data, and that the PEX server receiving the data decode it. This is transparent to the application developer or user, but can have some performance cost. By using the environment variable, you may regain this performance cost. Note that by default, this environment variable is not set on your machine.

Note – If you set the environment variable `SUNPEX_DIRECT_ENABLE` but are running your application remotely (that is, your application is running on a different machine than your server), the environment variable has no effect.

Features and Limitations of the Direct Rendering Path

Subset support

Direct PEXlib does not support any requests that are part of the Workstation subset of PEX.

Table 2-1 shows the return values for Direct PEXlib in the `PEXGetExtensionInfo` reply.

Table 2-1 PEXGetExtensionInfo reply (Direct PEXlib)

Field	Value
protocol major version	5
protocol minor version	1
vendor	"SunPEX 3.0 SMI (Direct)"
release number	30000
subset info	5 (PEXImmediateMode & PEXStructureMode)

Resource sharing

PEX resources are not shareable among different clients when using the Direct rendering path mode.

GDPs and GSEs

Solaris PEX PEXlib supports a large set of GDPs and GSEs. See Table 4-25 through Table 4-27 in Chapter 4, "Functional Description" for lists of the specific GDPs and GSEs supported in the current release of Solaris PEX.

Mixing X and PEX

For applications that mix X and PEX rendering between pairs of `PEXBeginRendering()` and `PEXEndRendering()` calls, make sure the Xlib function `XSync()` is called before switching from X to PEX, and vice versa. For better performance, applications should perform all X rendering before the `PEXBeginRendering()` or after the `PEXEndRendering()` to eliminate the overhead of the `XSync()` operation.

Mixing XGL and Direct PEXlib

If you mix XGL and PEXlib calls in the same application, you cannot run this application in the Direct rendering path mode.

Double Buffering with Escapes

Direct PEXlib supports the Evans & Sutherland and Hewlett-Packard double-buffering PEX extensions via the PEX Escape mechanism. For a list of these supported Escapes, see Table 4-28 in Chapter 4, “Functional Description.”

Direct PEXlib supports double buffering only on frame buffers that provide hardware double buffering, such as the GXplus. Refer to your frame buffer IHV for PEX double buffering support information.

Multibuffering

Direct PEXlib does not support interoperating with MBX (Multi-Buffering Extension to X).

Pixmaps

Direct PEXlib does not support rendering to pixmaps.

Extensions

Solaris PEX PEXlib provides header files containing constant definitions and data structures for extensions of Sun, Hewlett-Packard, and Evans & Sutherland such as GDPs, GSEs, and Escapes. These extension header files are:

- `ESpex.h`
- `ESproto.h`
- `ESprotost.h`
- `HPpex.h`
- `HPproto.h`
- `PEXExt.h`
- `PEXExtlib.h`
- `PEXExtproto.h`
- `PEXExtprotost.h`
- `SunPEX.h`
- `SunPEXproto.h`

References

- *PEXlib Programming Manual*
- *PEXlib Reference Manual*
- *PEXlib Specification and C Language Binding Version 5.1*
- *CGE PEX 5.1 Portability Guide*
- *Solaris PEXlib Reference Manual*

Server 3

The Solaris PEX server extension is an implementation of the X Consortium PEX 5.1protocol. It receives and interprets the PEX protocol. The Solaris PEX server is a display server that generates and controls the display on which your application draws 3D graphics.

Features

This section lists the major features for the Solaris PEX server. It also tells you where in this manual you can find examples of using the features.

▼ Subset support

Table 3-1 shows the return values for the Solaris PEX server in the `PEXGetExtensionInfo` reply

Note – The PHIGS Workstation subset is not supported in this release..

Table 3-1 PEXGetExtensionInfo reply (Server)

Field	Value
protocol major version	5
protocol minor version	1
vendor	“SunPEX 3.0 SMI”
release number	30000
subset info	5 (PEXImmediateMode & PEXStructureMode)

GDPs and GSEs

The Solaris PEX server supports a large set of GDPs and GSEs. See Table 4-25 through Table 4-27 in Chapter 4, “Functional Description” for lists of the specific 2D and 3D GDPs and GSEs supported in the current release of Solaris PEX.

Double Buffering with Escapes

The Solaris PEX server supports the Evans & Sutherland and Hewlett-Packard double buffering PEX extensions via the PEX Escape mechanism. See Table 4-28 in Chapter 4, “Functional Description” for the supported Escapes Enumerated Type Descriptors.

The Solaris PEX server supports double buffering only on frame buffers that provide hardware double buffering, such as the GXplus. Refer to your frame buffer IHV for PEX double buffering support information.

Note that if you try to request double buffering on a frame buffer that does not have hardware double buffering, the Solaris PEX server silently performs a single buffer.

Multibuffering

The Solaris PEX server does not support interoperating with MBX (Multi-Buffering Extension to X).

▼ Pixmaps

The Solaris PEX server does not support rendering to pixmaps.

Functional Description 4

This chapter lists the implementation-dependent functionality of Solaris PEX. It covers Enumerated Type Information, Implementation-Dependent Constants, PHIGS Workstation Dynamics, Lookup Tables, GSEs, GDPs, and Escapes.

Enumerated Type Information

Table 4-1 through Table 4-51 lists the enumerated types supported in this version of Solaris PEX. Each supported value is listed along with its mnemonic string and a brief description. The strings are returned exactly as shown.

Sun-Specific Enumerated Types

Sun-specific enumerated values always begin with `0x85` (bit `10000101`) in the first two bytes. CGE extended enumerated values always begin with `0x90` (bit `10010000`) in the first two bytes. The most significant bit (MSB) indicates that the enumerated type value is a vendor extension; that is, it is not part of the standard enumerated values defined by the X Consortium. The remaining bits of the first two bytes indicate which vendor has defined the value.

Table 4-1 PEXETMarkerType

Value	Mnemonic	Symbol	Description
1	Dot	.	The smallest displayable dot at the marker position. Ignore the marker_scale attribute.
2	Cross	+	A plus sign with intersection at the marker position.
3	Asterisk	❋	An asterisk with intersection at the marker position.
4	Circle	❍	An unfilled circle with center at the marker position.
5	X	✕	A multiplication sign with intersection at the marker position.
0x8500	SunSquare	❑	An unfilled square with center at the marker position.
0x8501	SunBowtieNE		An unfilled bowtie figure with intersection at the marker position and oriented with upper square in the upper right hand quadrant, lower square in the lower left hand quadrant.
0x8502	SunBowtieNW		An unfilled bowtie figure with intersection at the marker position and oriented with upper square in the upper left hand quadrant, lower square in lower right hand quadrant.

Table 4-2 PEXETATextStyle

Value	Mnemonic	Description
1	NotConnected	Draw the annotation text with no line connecting it to the reference point.
2	Connected	Draw the annotation text with a line connecting it to the reference point. Draw this line using the current line attributes.

Table 4-3 PEXETLineType

Value	Mnemonic	Description
1	Solid	Draw with a solid, unbroken line.
2	Dashed	Draw with a dashed line.
3	Dotted	Draw with a dotted line.
4	DashDot	Draw with a line of alternating dots and dashes.
0x8500	SunLongDash	Draw with a line of long dashes.
0x8501	SunDotDashDot	Draw with a line of dot followed by dash followed by dot, repeating.
0x8502	SunCenter	Draw with a line of a long dash followed by a short dash, repeating.
0x8503	SunPhantom	Draw with a line of a long dash followed by two short dashes, repeating.
0x9000	ExtLineTypeCenter	Draw with a line of a long dash followed by a short dash, repeating.
0x9001	ExtLineTypePhantom	Draw with a line of a long dash followed by two short dashes, repeating.

Table 4-4 PEXETPolylineInterpMethod

Value	Mnemonic	Description
1	None	Perform no color interpolation between polyline vertices. If the endpoints of a segment have different color values, Solaris PEX uses the color of the first vertex for drawing the entire segment.
2	Color	Linearly interpolate between the color values at the vertices, if any such colors were specified.

Table 4-5 PEXETCurveApproxMethod

Value	Mnemonic	Description
1	ImplementationDependent	This type is ConstantBetweenKnots (the same as value 2).
2	ConstantBetweenKnots	Tessellate the curve with equal parametric increments between successive pairs of knots. The value of *tolerance* controls tessellation of the curve. If the *tolerance* is not an integer value, truncate it and only use the integer portion. If *tolerance* is less than or equal to zero, evaluate the curve only at the parameter limits and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, evaluate the curve 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	Tessellate the curve until the length of each line segment (chord) in world coordinates is less than *tolerance*.
4	NPC_ChordalSize	Tessellate the curve until the length of each line segment (chord) in normalized projection coordinates is less than *tolerance*.
5	DC_ChordalSize	Tessellate the surface until the length of each line segment (chord) in device coordinates is less than *tolerance*.
6	WCS_ChordalDev	Tessellate the curve until the maximum deviation (in world coordinates) between the line and the curve is less than *tolerance*.
7	NPC_ChordalDev	Tessellate the curve until the maximum deviation (in normalized projection coordinates) between the line and the curve is less than *tolerance*.
8	DC_ChordalDev	Tessellate the curve until the maximum deviation (in device coordinates) between the line and the curve is less than *tolerance*.
9	WCS_Relative	Maintain a relative level of quality based on *tolerance* independent of scaling in world coordinates.
10	NPC_Relative	Maintain a relative level of quality based on *tolerance* independent of scaling in normalized projection coordinates.
11	DC_Relative	Maintain a relative level of quality based on *tolerance* independent of scaling in device coordinates.

Table 4-6 PEXETInteriorStyle

Value	Mnemonic	Description
1	Hollow	Do not fill the interiors of surface primitives, but draw the boundary using the surface color. If the surface primitive is clipped as a result of modeling, view, or workstation clipping, draw the boundary along the clipped boundary as well.
2	Solid	Fill the interiors of surface primitives using the surface color.
4	Hatch	Fill the interiors of the surface primitives using the surface color and the hatch style index specified by the interior style index.
5	Empty	Do not draw the interior of the surface primitive and the boundary at all.
0x9000	ExtInteriorStyleTexture	Fill the interiors of surface primitives using texture maps.

Currently, there are no X Consortium-wide registered hatch styles. Sun has defined the following 24 styles as listed in Table 4-7.

Table 4-7 PEXETHatchStyle

Value	Mnemonic	Description
0x8501	SunHorizontal	Horizontal lines
0x8533	SunHorizontalDouble	Double-spaced horizontal lines
0x8565	SunHorizontalTransparent	Transparent horizontal lines
0x8597	SunHorizontalDoubleTransparent	Double-spaced transparent horizontal lines
0x8502	SunVertical	Vertical lines
0x8534	SunVerticalDouble	Double-spaced vertical lines
0x8566	SunVerticalTransparent	Transparent vertical lines
0x8598	SunVerticalDoubleTransparent	Double-spaced transparent vertical lines
0x8503	SunDiagonal45	45-degree angled lines
0x8535	SunDiagonal45Double	Double-spaced 45-degree angled lines
0x8567	SunDiagonal45Transparent	Transparent 45-degree angled lines
0x8599	SunDiagonal45DoubleTransparent	Double-spaced transparent 45-degree angled lines
0x8504	SunDiagonal135	135-degree angled lines

Table 4-7 PEXETHatchStyle (Continued)

Value	Mnemonic	Description
0x8536	SunDiagonal135Double	Double-spaced 135-degree angled lines
0x8568	SunDiagonal135Transparent	Transparent 135-degree angled lines
0x859a	SunDiagonal135DoubleTransparent	Double-spaced transparent 135-degree angled lines
0x8505	SunGridRectangular	Rectangular grid
0x8537	SunGridRectangularDouble	Double-spaced rectangular grid
0x8569	SunGridRectangularTransparent	Transparent rectangular grid
0x859b	SunGridRectangularDoubleTransparent	Double-spaced transparent rectangular grid
0x8506	SunGridDiagonal	Diagonal grid
0x8538	SunGridDiagonalDouble	Double-spaced diagonal grid
0x856a	SunGridDiagonalTransparent	Transparent diagonal grid
0x859c	SunGridDiagonalDoubleTransparent	Double-spaced transparent diagonal grid
0x9000	ExtHatchStyle45Degrees	45-degree angled lines
0x9001	ExtHatchStyle135Degrees	135-degree angled lines

Table 4-8 PEXETSurfaceEdgeType

Value	Mnemonic	Description
1	Solid	Draw the surface edge with a solid, unbroken line.
2	Dashed	Draw the surface edge with a dashed line.
3	Dotted	Draw the surface edge with a dotted line.
4	DashDot	Draw the surface edge with a line of alternating dots and dashes.
0x8500	SunLongDash	Draw the surface edge with a line of long dashes.

Table 4-8 PEXETSurfaceEdgeType (Continued)

Value	Mnemonic	Description
0x8501	SunDotDashDot	Draw the surface edge with a line of dot followed by dash followed by dot, repeating.
0x8502	SunCenter	Draw the surface edge with a line of a long dash followed by a short dash, repeating.
0x8503	SunPhantom	Draw the surface edge with a line of a long dash followed by two short dashes, repeating.

Table 4-9 PEXETReflectionModel

Value	Mnemonic	Description
1	NoShading	Perform no light source shading computation. Light source illumination does not affect the surface color effectively, shaded color equals intrinsic color.
2	Ambient	Use only the ambient terms of the lighting equation. The shaded color will be the intrinsic color as seen under ambient light.
3	Diffuse	Use only the ambient and diffuse terms of the lighting equation. The shaded color will be the intrinsic color as seen under ambient light, plus a diffuse reflection component from each light source.
4	Specular	Use all of the ambient, diffuse, and specular terms of the lighting equation 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.

Table 4-10 PEXETSurfaceInterpMethod

Value	Mnemonic	Description
1	None	Use the color resulting from a single light source computation for the entire surface. Perform no interpolation across surface interiors or edges.
2	Color	Compute the colors at the vertices of the surface using the current reflection model. Then linearly interpolate these colors across the interior of the surface or along the edges.

Table 4-11 PEXETSurfaceApproxMethod

Value	Mnemonic	Description
1	ImplementationDependent	This type is ConstantBetweenKnots (the same as 2).
2	ConstantBetweenKnots	Tessellate the surface with equal parametric increments between successive pairs of knots. The two *tolerance* values control tessellation in each of the two parameter dimensions. If the *tolerance* values are not integer values, truncate them and use only the integer portions of each. If *u_tolerance* is less than or equal to zero, evaluate the surface 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, evaluate the surface 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. Use the value of *v_tolerance* similarly to control the evaluation in the *v* direction.
3	WCS_ChordalSize	Tessellate 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	Tessellate 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	Tessellate 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 world coordinates in the *v* parameter direction is less than the specified *v_tolerance* value
6	WCS_PlanarDev	Tessellate the surface into facets: subdivide the surface until the absolute value of the maximum deviation, in world coordinates, between any facet and the surface is less than *u_tolerance*. *v_tolerance* is ignored.
7	NPC_PlanarDev	Tessellate the surface into facets: subdivide 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*. *v_tolerance* is ignored.

Table 4-11 PEXETSurfaceApproxMethod (Continued)

8	DC_PlanarDev	Tessellate the surface into facets: subdivide the surface until the absolute value of the maximum deviation, in device coordinates, between any facet and the surface is less than *u_tolerance*. *v_tolerance* is ignored.
9	WCS_Relative	Maintain a relative level of quality based on *u_tolerance* independent of scaling in world coordinates. *v_tolerance* is ignored.
10	NPC_Relative	Maintain a relative level of quality based on *u_tolerance* independent of scaling in normalized projection coordinates. *v_tolerance* is ignored.
11	DC_Relative	Maintain a relative level of quality based on *u_tolerance* independent of scaling in device coordinates. *v_tolerance* is ignored.

Table 4-12 PEXETTrimCurveApproxMethod

Value	Mnemonic	Description
1	ImplementationDependent	This type is ConstantBetweenKnots (the same as 2).
2	ConstantBetweenKnots	Tessellate the trim curve with equal parametric increments between successive pairs of knots. The *tolerance* value controls tessellation of the trim curve. If *tolerance* is not an integer value, truncate it and use only the integer portion. If *tolerance* is less than or equal to zero, evaluate the trim curve only at the parameter limits and at the knots that are within the specified parameter range. If *tolerance* is greater than zero, evaluate the trim curve 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.

Table 4-13 PEXETModelClipOperator

Value	Mnemonic	Description
1	Replace	Use the specified half-spaces to create a new composite model clip volume to replace the current composite model clip volume.
2	Intersection	Intersect the specified half-spaces with the current composite model clip volume to compute a new composite model clip volume.

Table 4-14 PEXETColorType

Value	Mnemonic	Description
0	Indexed	A color that is passed as an unsigned 16-bit integer (that is, 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, 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: PEXFLOAT].
5	RGBInt8	A color that is passed as a unit of four bytes, in the order of 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 of red, green, blue. A color in this format has a type defined by COLOR_RGB_INT16:[r, g, b, pad: CARD16].

Table 4-15 PEXETFloatFormat

Value	Mnemonic	Description
1	IEEE_754_32	An IEEE 754 standard 32-bit floating-point value

Table 4-16 PEXETLightType

Value	Mnemonic	Description
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.

Table 4-16 PEXETLightType (Continued)

Value	Mnemonic	Description
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.

Table 4-17 PEXETHLHSRMode

Value	Mnemonic	Description
1	Off	Draw all output primitives in the order they are processed. Make no attempt to remove hidden surfaces.
2	ZBuffer	Resolve visibility at each pixel, using a depth- or z-buffering technique. The z-buffering method and number of bits of precision in the *z*-value are device-dependent.
6	ZBufferId	Same as ZBuffer, except that the HLHSRId is used to enable and disable z-buffering.

Table 4-18 PEXETPickDeviceType

Value	Mnemonic	Description
1	DC_HitBox	The picking aperture is defined by a pick position and a pick distance, both in device coordinates. The shape of the hit box (square, circle, and so forth) is implementation-dependent. The pick distance defines the half-width or radius of the hit box.
2	NPC_HitVolume	The picking aperture is defined by two points that define a hit volume in NPC space. Any primitives intersecting with the volume will be selected.

Table 4-19 PEXETColorApproxType

Value	Mnemonic	Description
1	ColorSpace	Convert the rendering pipeline color into a color with three individual color components.

Table 4-20 PEXETColorApproxModel

Value	Mnemonic	Description
1	RGB	red, green, blue

Table 4-21 PEXETRenderingColorModel

Value	Mnemonic	Description
0	ImplementationDependent	This type is RGB (the same as 1).
1	RGB	red, green, blue color model

Table 4-22 PEXETParametricSurfaceCharacteristics

Value	Mnemonic	Description
1	None	No additional surface characteristics beyond the current surface attributes.
2	ImplementationDependent	An implementation-dependent method that displays the shape of the surface. Do 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. How the representation interacts with any interior rendering indicated by the interior attributes is implementation-dependent. The data record is ignored for this type.

Table 4-22 PEXETParametricSurfaceCharacteristics (Continued)

Value	Mnemonic	Description
3	IsoparametricCurves	Draw isoparametric curves 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*, space the specified number of curves evenly between the parameter limits of the surface; also draw curves at the parameter limits. If the placement is *NonUniform*, space the specified number of curves evenly between each pair of knots; also draw curves at the knots. In both cases, only draw the portions of isoparametric curves that are within the interior of the surface as defined by any trimming curves. Do 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. Draw isoparametric curves 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.

Table 4-23 PEXETPickOneMethod

Value	Mnemonic	Description
1	Last	Return the last primitive picked.
3	VisibleAny	Return any picked primitive which is visible, considering the current HLHSR mode.

Table 4-24 PEXETPickAllMethod

Value	Mnemonic	Description
1	All	Return all primitives which are within or which intersect the pick aperture.
2	Visible	Return only the visible primitives which are within or which intersect the pick aperture.

Table 4-25 PEXETGDP2D

Value	Mnemonic	Description
0x8502	SunCircle	Circle 2D
0x8503	SunCircularArc	Circular Arc 2D
0x8504	SunCircularArcClose	Circular Arc Close 2D
0x8505	SunAnnotationCircle	Annotation Circle 2D
0x8506	SunAnnotationCircularArc	Annotation Circular Arc 2D
0x8507	SunAnnotationCircularArcClose	Annotation Circular Arc Close 2D
0x8508	SunEllipse	Ellipse 2D
0x8509	SunEllipticalArc	Elliptical Arc 2D
0x850a	SunEllipticalArcClose	Elliptical Arc Close 2D
0x850b	SunAnnotationEllipse	Annotation Ellipse 2D
0x850c	SunAnnotationEllipticalArc	Annotation Elliptical Arc 2D
0x850d	SunAnnotationEllipticalArcClose	Annotation Elliptical Arc Close 2D
0x850f	SunRectangularGrid	Rectangular Grid 2D
0x8510	SunRadialGrid	Radial Grid 2D

Table 4-26 PEXETGDP

Value	Mnemonic	Description
0x8502	SunCircle	Circle 3D
0x8503	SunCircularArc	Circular Arc 3D
0x8504	SunCircularArcClose	Circular Arc Close 3D
0x8505	SunAnnotationCircle	Annotation Circle 3D
0x8506	SunAnnotationCircularArc	Annotation Circular Arc 3D

Table 4-26 PEXETGDP

Value	Mnemonic	Description
0x8507	SunAnnotationCircularArcClose	Annotation Circular Arc Close 3D
0x8508	SunEllipse	Ellipse 3D
0x8509	SunEllipticalArc	Elliptical Arc 3D
0x8510	SunEllipticalArcClose	Elliptical Arc Close 3D
0x8511	SunAnnotationEllipse	Annotation Ellipse 3D
0x8512	SunAnnotationEllipticalArc	Annotation Elliptical Arc 3D
0x8513	SunAnnotationEllipticalArcClose	Annotation Elliptical Arc Close 3D
0x8514	SunRectangularGrid	Rectangular Grid 3D
0x8515	SunRadialGrid	Radial Grid 3D
0x8516	SunTriangleList	Triangle List 3D
0x8401	E&S_Spheres	List of Spheres
0x8402	E&S_Spheres_withRadii	List of Spheres with Radii
0x8403	E&S_Spheres_withColors	List of Spheres with Colors
0x8404	E&S_Spheres_withRadii&Colors	List of Spheres with Radii and Colors
0x8405	E&S_Cylinders	Lists of Cylinders
0x8406	E&S_Cylinders_withRadii	Lists of Cylinders with Radii
0x8407	E&S_Cylinders_withColors	Lists of Cylinders with Colors
0x8408	E&S_Cylinders_withRadii&Colors	Lists of Cylinders with Radii and Colors

Table 4-27 PEXETGSE

Value	Mnemonic	Description
0x8501	SunSetHighlightColor	Set Highlight Color
0x8504	SunSetTextSlantAngle	Set Text Slant Angle
0x8505	SunSetAnnotationTextSlantAngle	Set Annotation Text Slant Angle

Table 4-27 PEXETGSE

Value	Mnemonic	Description
0x8507	SunSetStrokeAntiAliasParameters	Set Stroke Anti-Aliasing Parameters
0x8509	SunSetEndcap	Set Endcap
0x850a	SunSetStrokeJoin	Set Stroke Join
0x850b	SunSetSilhouetteEdgeFlag	Set Silhouette Edge Flag
0x850c	SunSetSurfaceTransparencyCoefficient	Set Surface Transparency Coefficient
0x8401	E&S_Sphere_Radius	Sphere Radius
0x8402	E&S_Sphere_Precision	Sphere Divisions
0x8403	E&S_Cylinder_Radius	Cylinder Radius
0x8404	E&S_Cylinder_Precision	Cylinder Divisions

Table 4-28 PEXETEscape

Value	Mnemonic	Description
1	SetEchoColor	Set echo color
0x8401	ES_ESCAPE_DBLBUFFER	Set the specified Drawable to be double-buffered
0x8402	ES_ESCAPE_SWAPBUFFER	Swap buffers on the specified Drawable
0x8403	ES_ESCAPE_SWAPBUFFERCONTENT	Inquire the state of the specified double-buffered Drawable
0x8701	HP_ESCAPE_DFRONT	Set rendering buffer
0x8505	SunDefineMarkerType	User-definable markers
0x8506	SunGetMarkerDescription	Report user-definable marker
0x8507	SunChangeExtendedRendererAttributes	Change extended renderer attributes
0x8508	SunGetExtendedRendererAttributes	Report extended renderer attributes
0x8509	SunGetExtendedRendererAttributeDynamics	Report extended renderer attributes dynamics
0x8510	SunFlushRenderer	Flushes all pending output commands

Table 4-28 PEXETEscape

Value	Mnemonic	Description
0x9000	ExtEscapeChangePipelineContext	Set pipeline context extended
0x9001	ExtEscapeGetPipelineContext	Get pipeline context extended
0x9002	ExtEscapeChangeRenderer	Change renderer attributes extended
0x9003	ExtEscapeGetRendererAttributes	Get renderer attributes extended
0x9004	ExtEscapeSetTableEntries	Set table entries extended
0x9005	ExtEscapeGetTableEntries	Get table entries extended
0x9006	ExtEscapeGetTableEntry	Get table entry entended
0x9007	ExtEscapeCreateTM	Create texture map
0x9008	ExtEscapeCreateTMDescription	Create texture map description
0x9009	ExtEscapeFreeTM	Free texture map
0x900a	ExtEscapeFreeTMDescription	Free texture map description
0x900b	ExtEscapeFetchElements	Fetch elements extended
0x900c	ExtEscapeQueryColorApprox	Inquire supported color approximation
0x900d	ExtEscapeCreateTMExtraData	Create extra texture data

Table 4-29 PEXExtETEnumType

Value	Mnemonic	Description
0x9000	ExtEnumType	Extended enumerated types
0x9001	ExtOC	Extended output commands
0x9002	ExtPC	Extended pipeline context attributes
0x9003	ExtRA	Extended renderer attributes
0x9004	ExtLUT	Extended lookup tables
0x9005	ExtID	Extended implementation dependent constants
0x9006	ExtTMRenderingOrder	Extended texture mapping rendering order

Table 4-29 PEXExtETEnumType

Value	Mnemonic	Description
0x9007	ExtTMCoordSource	Extended texture mapping coordinate source
0x9008	ExtTMCompositeMethod	Extended texture mapping composition method
0x9009	ExtTMTexelSampleMethod	Extended texture mapping sample method
0x900a	ExtTMBoundaryCondition	Extended texture mapping boundary condition
0x900b	ExtTMClampColorSource	Extended texture mapping clamp color source
0x900c	ExtTMDomain	Extended texture mapping domain
0x900d	ExtTexelType	Extended texture mapping texel types
0x900e	ExtTMResourceHint	Extended texture mapping resource hints
0x900f	ExtTMType	Extended texture mapping types
0x9010	ExtTMParameterizationMethod	Extended texture mapping parameterization method
0x9011	ExtTMPerspectiveCorrection	Extended texture mapping perspective correction
0x9012	ExtTMSampleFrequency	Extended texture mapping sample frequency
0x9013	ExtPrimitiveAAMode	Extended primitive anti-aliasing mode
0x9014	ExtPrimitiveAABlendOp	Extended primitive anti-aliasing blending operation
0x9015	ExtLineCapStyle	Extended line cap style
0x9016	ExtLineJoinStyle	Extended line join style

Table 4-30 PEXExtETOC

Value	Mnemonic	Description
0x9000	ExtOCTMPerspectiveCorrection	Set texture mapping perspective correction
0x9001	ExtOCTMSampleFrequency	Set texture mapping sample frequency
0x9002	ExtOCTMResourceHints	Set texture mapping resource hints
0x9003	ExtOCActiveTextures	Set texture mapping active textures
0x9004	ExtOCBFActiveTextures	Set texture mapping backface active textures

Table 4-30 PEXExtETOC

Value	Mnemonic	Description
0x9005	ExtOCFillAreaSetWithData	Extended fill area set with data
0x9006	ExtOCSetOfFillAreaSets	Extended set of fill area sets with data
0x9007	ExtOCTriangleStrip	Extended triangle strip with data
0x9008	ExtOCQuadrilateralMesh	Extended quadrilateral mesh with data
0x9009	ExtOCPrimitiveAA	Set primitive anti-aliasing
0x900a	ExtOCLineCapStyle	Set line cap style
0x900b	ExtOCLineJoinStyle	Set line join style
0x900c	ExtOCEllipse	Ellipse
0x900d	ExtOCEllipse2D	2D ellipse
0x900e	ExtOCCircle2D	2D circle
0x900f	ExtOCEllipticalArc	Elliptical Arc
0x9010	ExtOCEllipticalArc2D	2D Elliptical Arc
0x9011	ExtOCCircularArc2D	2D Circular Arc

Table 4-31 PEXExtETPC

Value	Mnemonic	Description
0x9000	ExtPCTMPerspectiveCorrection	Texture mapping perspective correction method
0x9001	ExtPCTMResourceHints	Texture mapping resource hints
0x9002	ExtPCTMSampleFrequency	Texture mapping sample frequency
0x9003	ExtPCActiveTextures	Texture mapping active textures
0x9004	ExtPCBFActiveTextures	Texture mapping backface active textures
0x9005	ExtPCPrimitiveAA	Primitive anti-aliasing mode and blend function
0x9006	ExtPCLineCapStyle	Line cap style
0x9007	ExtPCLineJoinStyle	Line join style

Table 4-32 PEXExtETRA

Value	Mnemonic	Description
0x9000	ExtRATMBindingTable	Texture mapping binding table renderer attribute
0x9001	ExtRATMCoordSourceTable	Texture mapping coordinate source table renderer attribute
0x9002	ExtRATMCompositionTable	Texture mapping composition table renderer attribute
0x9003	ExtRATMSamplingTable	Texture mapping sampling table renderer attribute

Table 4-33 PEXExtETLUT

Value	Mnemonic	Description
0x9000	ExtLUTTMBinding	Texture mapping binding table
0x9001	ExtLUTTMCoordSource	Texture mapping coordinate source table
0x9002	ExtLUTTMComposition	Texture mapping composition table
0x9003	ExtLUTTMSampling	Texture mapping sampling table

Table 4-34 PEXExtETID

Value	Mnemonic	Description
0x9000	ExtIDMaxTextureMaps	Maximum simultaneously active texture maps per primitive
0x9001	ExtIDMaxFastTMSize	Maximum size of any dimension of the base level of a texture map which allows an optimized implementation
0x9002	ExtIDPowerOfTwoTMSizesRequired	Indicates whether all texel arrays defining a texture must be powers of two.
0x9003	ExtIDSquareTMRequired	Indicates whether a texture map must have equally sized dimensions.

Table 4-35 PEXExtETTMRenderingOrder

Value	Mnemonic	Description
0x9000	ExtTMRenderingOrderPreSpecular	The texture map is to be applied to the primitive before the speculation reflection calculations.
0x9001	ExtTMRenderingOrderPostSpecular	The texture map is to be applied to the primitive after the speculation reflection calculations.

Table 4-36 PEXExtETTMCoordSource

Value	Mnemonic	Description
0x9000	ExtTMCoordSourceVertexCoord	The source texture coordinate values are the vertex coordinate.
0x9001	ExtTMCoordSourceVertexNormal	The source texture coordinate values are the vertex normal.
0x9002	ExtTMCoordSourceFloatData	The source texture coordinate values are the vertex's floating point data list.

Table 4-37 PEXExtETTMCompositeMethod

Value	Mnemonic	Description
0x9000	ExtTMCompositeReplace	Texture map data replaces the primitive's current data.
0x9001	ExtTMCompositeModulate	Texture map data and primitive data are blended.
0x9002	ExtTMCompositeBlendEnvColor	The primitive color and a constant environment color are blended by the texture map color.
0x9003	ExtTMCompositeDecal	Texture map color and primitive color are blended by the texture map alpha.
0x9004	ExtTMCompositeDecalBackground	Texture map color and background color are blended according to the texture map alpha.
0x9005	ExtTMCompositeReplaceBlendedColors	The primitive color is replaced by two constant colors modulated by the texture color.

Table 4-38 PEXExtETTMTexelSampleMethod

Value	Mnemonic	Description
0x9000	ExtTMTexelSampleSingleBase	The closest single texel selected by the texture coordinates is sampled from the base texture map level.
0x9001	ExtTMTexelSampleLinearBase	The neighboring texels selected by the texture coordinate(s) in the base texture map level are weighted average.
0x9002	ExtTMTexelSampleSingleInMipmap	The closest single texel selected by the texture coordinate(s) is sampled from the closest texture map level to the sample depth.
0x9003	ExtTMTexelSampleLinearInMipmap	The neighboring texels selected by the texture coordinate(s) are sampled from the closest texture map level to the sample depth. Their weighted average is used.
0x9004	ExtTMTexelSampleSingleBetweenMipmaps	The closest texel selected by the texture coordinate(s) is sampled from the two closest texture map levels to the sample depth. The texel found at the exact sample depth by linear interpolation between these two sampled texels is used.
0x9005	ExtTMTexelSampleLinearBetweenMipmaps	The neighboring texels selected by the texture coordinate(s) are sampled from the two closest texture map levels to the sample depth. A weighted average is taken of those texels on each of the texture map levels.

Table 4-39 PEXExtETTMBoundaryCondition

Value	Mnemonic	Description
0x9000	ExtTMBoundaryCondClampColor	A color specified by the clamp color source is applied.
0x9001	ExtTMBoundaryCondBoundary	Texture map boundary color is applied.
0x9002	ExtTMBoundaryCondWrap	Texel sampling wraps back to the opposite texture map border.
0x9003	ExtTMBoundaryCondMirror	Texel sampling is reversed across the texture map.

Table 4-40 PEXExtETTMClampColorSource

Value	Mnemonic	Description
0x9000	ExtTMClampColorSourceAbsolute	Primitive intrinsic color is applied.
0x9001	ExtTMClampColorSourceExplicit	The explicitly specified color is used.

Table 4-41 PEXExtETTMDomain

Value	Mnemonic	Description
0x9000	ExtTMDomainColor1D	1D texture map
0x9001	ExtTMDomainColor2D	2D texture map

Table 4-42 PEXExtETTexelType

Value	Mnemonic	Description
0x9000	ExtTexelLuminanceFloat	A texel which is passed as a floating point value [0.0 -> 1.0] representing an intensity
0x9001	ExtTexelLuminanceInt8	A texel which is passed as an unsigned 8-bit integer [0 -> 255] representing an intensity
0x9002	ExtTexelLuminanceInt16	A texel which is passed as an unsigned 16-bit integer [0 -> 65535] representing an intensity
0x9003	ExtTexelLuminanceAlphaFloat	A texel which is passed as two floating point values [0.0 -> 0.1] representing an intensity and an alpha value
0x9004	ExtTexelLuminanceAlphaInt8	A texel which is passed as two unsigned 8-bit integers [0 -> 255] representing an intensity and an alpha value
0x9005	ExtTexelLuminanceAlphaInt16	A texel which is passed as two unsigned 16-bit integers [0 -> 65535] representing an intensity and an alpha value
0x9006	ExtTexelRGBFloat	A texel which is passed as three floating point values [0.0 -> 1.0] representing the red, green, and blue components of a color

Table 4-42 PEXExtETTexelType (Continued)

Value	Mnemonic	Description
0x9007	ExtTexelRGBInt8	A texel which is passed as three unsigned 8-bit integers [0 -> 255] representing the red, green, and blue components of a color
0x9008	ExtTexelRGBInt16	A texel which is passed as three unsigned 16-bit integers [0 -> 65535] representing the red, green, and blue components of a color
0x9009	ExtTexelRGBAlphaFloat	A texel which is passed as four floating point values [0.0 -> 0.1] representing the red, green, and blue components of a color and an alpha value
0x9010	ExtTexelRGBAlphaInt8	A texel which is passed as four unsigned 8-bit integers [0 -> 255] representing the red, green, and blue components of a color and an alpha value
0x9011	ExtTexelRGBAlphaInt16	A texel which is passed as four unsigned 16-bit integers [0 -> 65535] representing the red, green, and blue components of a color and an alpha value

Table 4-43 PEXExtETTMResourceHint

Value	Mnemonic	Description
0x9000	ExtTMResourceHintNone	No optimization considerations in particular are made.
0x9001	ExtTMResourceHintSpeed	Optimize for speed consideration.
0x9002	ExtTMResourceHintSpace	Optimize for space consideration.

Table 4-44 PEXExtETTMType

Value	Mnemonic	Description
0x9000	ExtTMTypeMipMap	Mipmap texture map

Table 4-45 PEXExtETTMParameterizationMethod

Value	Mnemonic	Description
0x9000	ExtTMParamExplicit	The primitive's texture coordinates are explicitly defined by the texture coordinate source in the current texture coordinate source lookup table.
0x9001	ExtTMParamReflectSphereVRC	The source texture map coordinates are a reflection vector which is normalized and conceptually projected onto an infinite sphere (with (0,0,0) as its origin) projection object. Calculations are performed in VRC space after the vertex and its normal have been transformed to VRC space.
0x9002	ExtTMParamReflectSphereWC	The source texture map coordinates are a reflection vector which is normalized and conceptually projected onto an infinite sphere (with (0,0,0) as its origin) projection object. Calculations are performed in WC space after the vertex and its normal have been transformed to WC space.
0x9003	ExtTMParamLinearVRC	The vertex coordinates are transformed to VRC by using the current composite model transform and the current view orientation transform. The transformed vertex coordinates are then used to linearly project the values.

Table 4-46 PEXExtETTMPerspectiveCorrection

Value	Mnemonic	Description
0x9000	ExtTMPerspCorrectNone	No perspective correction
0x9002	ExtTMPerspCorrectPixel	Perspection correction is applied per pixel

Table 4-47 PEXExtETTMSampleFrequency

Value	Mnemonic	Description
0x9000	ExtTMSampleFrequencyPixel	Texture map texels are sampled once for each pixel.

Table 4-48 PEXExtETPrimitiveAAMode

Value	Mnemonic	Description
0x9000	ExtPrimAANone	No anti-aliasing.
0x9001	ExtPrimAAPoint	Aliasing corrections are made to “point” primitives.
0x9002	ExtPrimAAVector	Aliasing corrections are made to “vector” primitives.
0x9003	ExtPrimAAPointVector	Aliasing corrections are made to “point” and “vector” primitives.
0x9004	ExtPrimAAPolygon	Aliasing corrections are made to “polygon” primitives.
0x9005	ExtPrimAAPointPolygon	Aliasing corrections are made to “point” and “polygon” primitives.
0x9006	ExtPrimAAVectorPolygon	Aliasing corrections are made to “vector” and “polygon” primitives.
0x9007	ExtPrimAAPointVectorPolygon	Aliasing corrections are made to “point”, “vector”, and “polygon” primitives.

Table 4-49 PEXExtETPrimitiveAABlendOp

Value	Mnemonic	Description
0x9000	ExtPrimAABlendOpImpDep	The blending function is calculated based on surface transparency coefficient.

Table 4-50 PEXExtETLineCapStyle

Value	Mnemonic	Description
0x9000	ExtLineCapStyleButt	The wide line is square at the endpoint with no projection beyond.
0x9001	ExtLineCapStyleRound	The wide line has a circular arc with the diameter equal to the linewidth, centered on the endpoint.
0x9002	ExtLineCapStyleProjecting	The wide line is square at the end, but the path continues beyond the endpoint for a distance equal to half the linewidth.

Table 4-51 PEXExtETLineJoinStyle

Value	Mnemonic	Description
0x9000	ExtLineJoinStyleImpDep	The wide line segments are joined in a device-dependent fashion.
0x9001	ExtLineJoinStyleRound	Two line segments are joined by rendering a circular arc whose diameter is equal to the linewidth and whose centerpoint is the vertex between the two segments.
0x9002	ExtLineJoinStyleMiter	The outer edges of the wide line segments extend to meet at an angle.
0x9003	ExtLineJoinStyleBevel	The two line segments are joined by applying the butt cap style to each of them and rendering the triangular notch left between them.

Implementation-Dependent Constants

Table 4-52 lists the values of the implementation-independent constants in Solaris PEX.

Table 4-52 Implementation-Dependent Constants

Name	**Type**	**Value**
NomLineWidth	CARD32	1
NumSupportedLineWidths	CARD32	0[1]
MinLineWidth	CARD32	1
MaxLineWidth	CARD32	1000
NomEdgeWidth	CARD32	1
NumSupportedEdgeWidths	CARD32	0[1]
MinEdgeWidth	CARD32	1
MaxEdgeWidth	CARD32	1000
NomMarkerSize	CARD32	11
NumSupportedMarkerSizes	CARD32	0[1]
MinMarkerSize	CARD32	1
MaxMarkerSize	CARD32	MAXINT
MaxNameSetNames	CARD32	256
MaxModelClipPlanes	CARD32	10
TransparencySupported	CARD32	1 (TRUE)
Dithering Support	CARD32	hardware dependent
MaxNonAmbientLights	CARD32	8
MaxNURBOrder	CARD32	10
MaxTrimCurveOrder	CARD32	6
BestColorApproxValues	CARD32	PEXColorApproxAnyValues
DoubleBufferingSupported	CARD32	hardware dependent[2]

Table 4-52 Implementation-Dependent Constants (Continued)

Name	Type	Value
MaxHitsEventSupported	CARD32	0 (FALSE)
ChromaticityRedU	PEXFLOAT	0.628
ChromaticityRedV	PEXFLOAT	0.346
LuminanceRed	PEXFLOAT	1.0
ChromaticityGreenU	PEXFLOAT	0.268
ChromaticityGreenV	PEXFLOAT	0.588
LuminanceGreen	PEXFLOAT	1.0
ChromaticityBlueU	PEXFLOAT	0.150
ChromaticityBlueV	PEXFLOAT	0.070
LuminanceBlue	PEXFLOAT	1.0
ChromaticityWhiteU	PEXFLOAT	0.313
ChromaticityWhiteV	PEXFLOAT	0.329
LuminanceWhite	PEXFLOAT	1.0

1. A value of 0 indicates a continuous range of values are supported between the minimum and maximum values for these attributes.
2. See “Double Buffering with Escapes” on page 10 for details.

Lookup Tables

Table 4-53 lists the maximum number of entries definable for each lookup table. Table 4-54 through Table 4-64 lists the predefined entries for each lookup table type.

Table 4-53 Maximum Definable Entries for Lookup Table Types

Name	Maximum Number of Entries
LineBundle	1024
MarkerBundle	1024
TextBundle	1024
InteriorBundle	1024
EdgeBundle	1024
Pattern	0[1]
TextFont	1024
Color	256
View	64
Light	1024
DepthCue	1024
ColorApprox	32
TMBinding[2]	32
TMCoordSource[2]	32
TMComposition[2]	32
TMSampling[2]	32

1. Pattern lookup tables are not supported in Solaris PEX.

2. These lookup tables are defined as part of the COSE CGE PEX 5.1 extensions.

Table 4-54 Predefined Line Bundle Entries

Index	Line type	Interpolation method	Color	Line width scale factor	Approximation type	Approximation value
1	Solid	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
2	Dashed	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
3	Dotted	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
4	Dash-dot	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
5	Long-dash	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
6	Dot-dash-dot	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
7	Center	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0
8	Phantom	None	(*Indexed*, 1)	1.0	implementation-dependent	1.0

Table 4-55 Predefined Marker Bundle Entry

Index	Marker type	Marker size scale factor	Color
1	Asterisk	1.0	(*Indexed*, 1)

Table 4-56 Predefined Text Bundle Entries

Index	Font Index	Text Precision	Expansion Factor	Character Spacing	Color
1	1	Stroke	1.0	0.0	(*Indexed*, 1)

Table 4-57 Predefined Text Font Bundle Entries

Index	Value	Explanation
1	Roman.M	Each Text Font Lookup Table entry is a list of font identifiers. The Solaris PEX server allows a maximum of 64 font identifiers per entry and sets all 64 font ids of this predefined entry to a server-generated id that corresponds to the default font *Roman_M*.

Table 4-58 Predefined Interior Bundle Entry

Index	Attribute	Value
1	interior_style	1 (hollow)
	interior_style_index	1
	reflection_model	1 (none)
	surface_interp	1 (none)
	bf_interior_style	1 (hollow)
	bf_interior_style_index	1
	bf_reflection_model	1 (none)
	bf_surface_interp	1 (none)
	surface_approx	{1, 1.0, 1.0}
	surface_color	{*Indexed*, 1}
	reflection_attr	{1.0, 1.0, 1.0, 0.0, 0.0, (*Indexed*, 1)]
	bf_surface_color	{*Indexed*, 1}
	bf_reflection_attr	{1.0, 1.0, 1.0, 0.0, 0.0, (*Indexed*, 1)]

Table 4-59 Predefined Depth Cue Bundle Entries

Index	Depth Cue Mode	Depth Cue Reference Planes	Depth Cue Scale Factors	Color
0	PEXOff	(0.0, 1.0)	(1.0, 1.0)	(*Indexed*, 0)
1	PEXOn	(0.0, 1.0)	(0.0, 1.0)	(*Indexed*, 0)

Table 4-60 Predefined Edge Bundle Entries

Index	Edge flag	Edge type	Edge width scale factor	Color
1	Off	Solid	1.0	(*Indexed*, 1)
2	Off	Dashed	1.0	(*Indexed*, 1)
3	Off	Dotted	1.0	(*Indexed*, 1)
4	Off	Dash-dot	1.0	(*Indexed*, 1)
5	Off	Long-dash	1.0	(*Indexed*, 1)
6	Off	Dot-dash-dot	1.0	(*Indexed*, 1)
7	Off	Center	1.0	(*Indexed*, 1)
8	Off	Phantom	1.0	(*Indexed*, 1)

Table 4-61 Predefined Light Bundle Entries

Index	Type	Color	Direction
1	Ambient	(*Indexed*, 1)	None
2	Directional	(*Indexed*, 1)	[1.0, 1.0, 1.0]
3	Directional	(*Indexed*, 1)	[-1.0, 1.0, 1.0]
4	Directional	(*Indexed*, 1)	[1.0, -1.0, 1.0]
5	Directional	(*Indexed*, 1)	[1.0, 1.0, -1.0]

Table 4-62 Predefined View Entries

Index	Orientation Matrix	Mapping Matrix	Clip Limits	Clip Flag
0 (full view)	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$	ClipAll
1 (top view)	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.0 & 0.0 & 0.25 \\ 0.0 & 0.1 & 0.0 & 0.75 \\ 0.0 & 0.0 & 0.25 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0.45 \end{bmatrix}$ $\begin{bmatrix} 0.55 & 0.95 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$	ClipAll
2 (off axis view)	$\begin{bmatrix} 0.7071 & 0.0 & -0.7071 & 0.0 \\ -0.4083 & 0.8165 & -0.4083 & 0.0 \\ 0.5774 & 0.5774 & 0.5774 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.0 & 0.0 & 0.75 \\ 0.0 & 0.1 & 0.0 & 0.75 \\ 0.0 & 0.0 & 0.25 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.55 & 0.95 \end{bmatrix}$ $\begin{bmatrix} 0.55 & 0.95 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$	ClipAll
3 (front view)	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.0 & 0.0 & 0.25 \\ 0.0 & 0.1 & 0.0 & 0.25 \\ 0.0 & 0.0 & 0.25 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0.45 \end{bmatrix}$ $\begin{bmatrix} 0.05 & 0.45 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$	ClipAll
4 (right side view)	$\begin{bmatrix} 0.0 & 0.0 & -1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.0 & 0.0 & 0.75 \\ 0.0 & 0.1 & 0.0 & 0.25 \\ 0.0 & 0.0 & 0.25 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.55 & 0.95 \end{bmatrix}$ $\begin{bmatrix} 0.05 & 0.45 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$	ClipAll

Table 4-62 Predefined View Entries (Continued)

Index	Orientation Matrix	Mapping Matrix	Clip Limits	Clip Flag
5 (bottom view)	$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.1 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.1 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.25 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$	$\begin{bmatrix} 0.3 & 0.3 \end{bmatrix}$ $\begin{bmatrix} 0.7 & 0.7 \end{bmatrix}$ $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$	ClipAll

Table 4-63 Predefined Color Entries

Color Index	Color Type	Red	Green	Blue	Description
0	RGB	0.0	0.0	0.0	Black
1	RGB	1.0	1.0	1.0	White
2	RGB	1.0	0.0	0.0	Red
3	RGB	0.0	1.0	0.0	Green
4	RGB	0.0	0.0	1.0	Blue
5	RGB	1.0	1.0	0.0	Yellow
6	RGB	0.0	1.0	1.0	Cyan
7	RGB	1.0	0.0	1.0	Magenta

Table 4-64 Predefined Color Approximation Entries

Index	Type	Model	Maximum	Dither	Multipliers	Weights	Base Pixel
0	ColorSpace	RGB	5, 5, 5,	0 (off)	1, 6, 36	1.0, 1.0, 1.0	16

Common Graphics Environment (CGE) Extensions 5

This chapter contains all of the reference pages for the Sun supported CGE Extensions. Each utility function can be identified by its name in the NAME section at the top of each reference page.

▼ Extended Change Pipeline Context

NAME:

PEXExtChangePipelineContext - Extended Change Pipeline Context

SYNTAX:

void PEXExtChangePipelineContext (Display **display*, PEXPipelineContext *context*, unsigned long **value_mask*, PEXExtPCAttributes **values*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
context	The resource identifier of the extended pipeline context.
value_mask	A pointer to an array of three unsigned long.
values	A pointer to new values for attributes in the extended pipeline context.

RETURNS:

None

DESCRIPTION:

This function changes components of the specified extended pipeline context to the values specified. The value mask indicates which attribute values are specified. `PEXSetPCAttributeMaskAll` can be called to set up the non-extended portion of the value mask and `PEXExtSetPCAttributeMask` can be called to set up any portion of the value mask.

DATA STRUCTURES:

```
typedef XID        PEXPipelineContext;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXColorType
The specified color type is invalid or unsupported.

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXNameSet
The specified name set resource identified is invalid.

BadPEXPipelineContext
The specified pipeline context resource identifier is invalid.

BadValue
A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXCreatePipelineContext`, `PEXExtGetPipelineContext`, `PEXExtSetPCAttributeMask`, `PEXExtSetPCAttributeMaskAll`

Extended Change Renderer Attributes

NAME:

PEXExtChangeRenderer - Extended Change Renderer Attributes

SYNTAX:

void PEXExtChangeRenderer(Display **display*, PEXRenderer *renderer*,
unsigned long **value_mask*, PEXExtRendererAttributes **values*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
renderer	The resource identifier of the extended renderer.
value_mask	A pointer to an array of two unsigned long.
values	A pointer to new values for attributes in the extended renderer.

RETURNS:

None

DESCRIPTION:

This function changes attributes of the specified extended renderer to the values specified. The value mask indicates which attribute values are specified. `PEXExtSetRendererAttributeMaskAll` and `PEXExtSetRendererAttributeMask` can be called to set up the extended value mask.

DATA STRUCTURES:

```
typedef XID        PEXRenderer;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadMatch
The specified lookup table resource was not created with a drawable compatible with the drawable used to create the renderer resource.

BadPEXLookupTable
The specified lookup table resource identifier is invalid.

BadPEXNameSet
: The specified name set resource identified is invalid.

BadPEXPipelineContext
: The specified pipeline context resource identifier is invalid.

BadPEXRenderer
: The specified renderer resource identifier is invalid.

BadValue
: A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXCreateRenderer`, `PEXExtGetRendererAttributes`, `PEXExtSetRendererAttributeMask`, `PEXExtSetRendererAttributeMaskAll`, `PEXGetRendererDynamics`

▼ Extended Set Lookup Table Entries

NAME:

PEXExtSetTableEntries - Extended Set Lookup Table Entries

SYNTAX:

void PEXExtSetTableEntries(Display **display*, PEXLookupTable *table*,
unsigned int *start*, unsigned int *count*, int *table_type*, PEXPointer *entries*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
table	The resource identifier of the lookup table.
start	The index of the first table entry to be set.
count	The number of the table entries to be set.
table_type	The type of lookup table entries to be set.
entries	An array of table entries.

RETURNS:

None

DESCRIPTION:

The function sets the lookup table entries in the specified extended lookup table, starting at the specified entry index.

The entries must point to an array of structures having one of the following types:

`PEXTextFontEntry` if *table_type* is `PEXLUTTextFont` <5.1 types>
...
...
`PEXPatternEntry` if *table_type* is `PEXLUTPattern`
`PEXExtTMBindingEntry` if *table_type* is `PEXExtLUTTMBinding`
<extended types>
`PEXExtTMCoordSourceEntry` if *table_type* is `PEXExtLUTTMCoordSource`
`PEXExtTMCompositionEntry` if *table_type* is `PEXExtLUTTMComposition`
`PEXExtTMSamplingEntry` if *table_type* is `PEXExtLUTTMSampling`

DATA STRUCTURES:

```
typedef XID        PEXLookupTable;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadAlloc
The server failed to allocate the resource.

BadPEXColorType
The specified color type is invalid or unsupported.

BadPEXLookupTable
The specified lookup table resource identifier is invalid.

BadValue
A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXCreateRenderer`, `PEXExtGetRendererAttributes`, `PEXExtSetRendererAttributeMask`, `PEXExtSetRendererAttributeMaskAll`, `PEXGetRendererDynamics`

▼ Extended Create Texture Map

NAME:

PEXExtCreateTM - Create a Texture Map

SYNTAX:

PEXExtTextureMap PEXExtCreateTM(Display **display*, int *domain*,
PEXExtTMDomainData **domain_data*, PEXExtTexelArray **texel_arrays*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
domain	The texture map domain type.
domain_data	A pointer to domain information specifying the type of texture map to be created and the type of texel data.
texel_arrays	The texel data.

RETURNS:

The X resource identifier of the newly-created texture map.

DESCRIPTION:

This function converts the data described by the domain and texel array into an internal texture map resource. Once this function returns from its storage operation, all memory utilized by the texel arrays may be reclaimed for other use.

The *domain* and *domain_data* provide information global to the texture map to be created. The list of texel arrays provides the actual texel data. The number of arrays is determined by the number of levels and the texture map type. Texel arrays are ordered sequentially by logical levels, the base level being first in the list. Every subsequent map is ordered by its level from the largest dimension (t0, t1, or t2) down to the smallest dimension until all arrays have been defined. The texels are assumed to be stored in t0-t1-t2 order, that is t0 varies the fastest as texels are stored into the texel array.

DATA STRUCTURES:

```
typedef XID        PEXExtTextureMap;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadAlloc
: The server failed to allocate the resource.

BadIDChoice
: The specified ID already in use or not in range assigned to client.

BadMatch
: The number of texel arrays given does not match the number of texel arrays expected.

BadPEXFloatingPointFormat
: The specified floating point format is invalid or unsupported.

BadValue
: A specified value is out of range.

SEE ALSO:

`PEXExtCreateFilteredTM`, `PEXExtCreateFilteredTMFromWindow`, `PEXExtCreateTMDescription`

Extended Create Texture Map Description

NAME:

PEXExtCreateTMDescription - Create a Texture Map Description

SYNTAX:

PEXExtTMDescription PEXExtCreateTMDescription(Display **display*,
int *parameterization*, PEXExtTMParameterizationData **param_data*,
int *tm_rendering_order*, unsigned int *count*, PEXExtTextureMap **tm_ids*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
parameterization	The texture map parameterization type.
param_data	A pointer to parameterization data.
tm_rendering_order	The texture mapping rendering order (`PEXExtTMRenderingOrderPreSpecular`, `PEXExtTMRenderingOrderPostSpecular`).
count	The number of texture map resource identifiers.
tm_ids	A pointer to a list of texture map resource identifiers.

RETURNS:

The X resource identifier of the newly-created texture map description.

DESCRIPTION:

This function converts the data described by the parameterization and texture map resources into an internal texture map description resource. Once this function returns from its storage operation, all memory may be reclaimed for other use.

DATA STRUCTURES:

```
typedef XID        PEXExtTMDescription;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadAlloc
The server failed to allocate the resource.

BadIDChoice
The specified ID already in use or not in range assigned to client.

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadValue
A specified value is out of range.

SEE ALSO:

`PEXExtCreateTM`, `PEXExtCreateFilteredTM`,
`PEXExtCreateFilteredTMFromWindow`

Extended Free Texture Map

NAME:

PEXExtFreeTM - Free a Texture Map Resource

SYNTAX:

void PEXExtFreeTM(Display **display*, PEXExtTextureMap *texture_map*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
texture_map	The resource identifier of the texture map.

RETURNS:

None

DESCRIPTION:

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

DATA STRUCTURES:

```
typedef XID        PEXExtTextureMap;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadValue
The specified texture map ID is invalid or is an ID of a resource other than a texture map.

SEE ALSO:

`PEXExtCreateTM`

▼ Extended Free Texture Map Description

NAME:

PEXExtFreeTMDescription - Free a Texture Map Description Resource

SYNTAX:

void PEXExtFreeTMDescription(Display **display*, PEXExtTMDescription *tm_description*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
tm_description	The resource identifier of the texture map description.

RETURNS:

None

DESCRIPTION:

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

DATA STRUCTURES:

```
typedef XID        PEXExtTMDescription;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadValue
The specified *tm_description* is an invalid texture map description ID or is an ID of a resource other than a texture map description.

SEE ALSO:

`PEXExtCreateTMDescription`

Extended Get Pipeline Context Attributes

NAME:

PEXExtGetPipelineContext - Get Extended Pipeline Context Attributes

SYNTAX:

PEXExtPCAttributes *PEXExtGetPipelineContext(Display **display*,
PEXPipelineContext *context*, unsigned long **value_mask*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
context	The resource identifier of the extended pipeline context.
value_mask	A pointer to an array of three unsigned long.

RETURNS:

A pointer to the extended pipeline context values; a null pointer if unsuccessful.

DESCRIPTION:

This function returns the requested attributes values of the extended pipeline context. The value mask indicates which attribute values are requested. `PEXSetPCAttributeMaskAll` can be called to set up the non-extended portion of the value mask and `PEXExtSetPCAttributeMask` can be called to set up any portion of the value mask. PEXlib allocates the memory for the return value. `PEXExtFreePCAttributes` should be called to deallocate the memory.

DATA STRUCTURES:

```
typedef XID       PEXPipelineContext;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXPipelineContext
The specified pipeline context resource identifier is invalid.

BadValue

A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXCreatePipelineContext`, `PEXExtChangePipelineContext`, `PEXExtFreePCAttributes`

Extended Get Renderer Attributes

NAME:

PEXExtGetRendererAttributes - Get Extended Renderer Attribute Values

SYNTAX:

PEXExtRendererAttributes *PEXExtGetRendererAttributes(Display **display*,
PEXRenderer *renderer*, unsigned long **value_mask*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
renderer	The resource identifier of the extended renderer.
value_mask	A pointer to an array of two unsigned long. Indicates which attributes are to be returned from the renderer.

RETURNS:

A pointer to the extended renderer attribute values; a null pointer if unsuccessful.

DESCRIPTION:

This function returns the requested attributes values of the extended renderer. The value mask indicates which attribute values are requested. `PEXExtSetRendererAttributeMaskAll` and `PEXExtSetRendererAttributeMask` can be called to set up the extended value mask. PEXlib allocates the memory for the return value. `PEXExtFreeRendererAttributes` should be called to deallocate the memory.

DATA STRUCTURES:

```
typedef XID        PEXRenderer;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadValue

A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXCreateRenderer`, `PEXExtChangeRenderer`, `PEXGetRendererDynamics`, `PEXExtFreeRendererAttributes`

▼ Extended Get Table Entry

NAME:

PEXExtGetTableEntry - Extended Get Table Entry

SYNTAX:

PEXPointer PEXExtGetTableEntry(Display **display*, PEXLookupTable *table*,
    unsigned int *index*, int *value_type*, int **status_return*,
    int **table_type_return*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
table	The resource identifier of the lookup table.
index	The index of the entry to be returned from the lookup table.
value_type	The type of values to return (`PEXSetValue` or `PEXRealizedValue`).
status_return	Returns the entry status (`PEXDefinedEntry` or `PEXDefaultEntry`).
table_type_return	Returns the type of table.

RETURNS:

A pointer to the lookup table entry; a null pointer if unsuccessful.

DESCRIPTION:

This function returns a single entry from the specified lookup table. If the specified lookup table entry is defined, the defined entry is returned. If the specified lookup table entry is not defined, the default entry for that type of table is returned.

The type of structure returned depends on the specified type.

If the requested value type is `PEXSetValue`, the values returned will be those originally set in the table entry. If the requested value type is `PEXRealizedValue`, the values returned will be those actually used during rendering (i.e. the values used if the specified entry value can not be exactly matched).

PEXlib allocates memory for the returned entry. `PEXFreeTableEntries` should be called to deallocated the memory.

DATA STRUCTURES:

```
typedef XID        PEXLookupTable;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXLookupTable
The specified lookup table resource identifier is invalid.

BadValue
A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXExtSetTableEntries`, `PEXSetTableEntries`, `PEXFreeTableEntries`

Extended Get Table Entries

NAME:

PEXExtGetTableEntries - Extended Get Table Entries

SYNTAX:

Status PEXExtGetTableEntries(Display **display*, PEXLookupTable *table*,
unsigned int *start*, unsigned int *count*, int *value_type*,
int **table_type_return*, PEXPointer **entries_return*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
table	The resource identifier of the lookup table.
start	The index of the first lookup table entry to be returned.
count	The number of entries requested.
value_type	The type of values to return (`PEXSetValue` or `PEXRealizedValue`).
table_type_return	Returns the type of table.
entries_return	Returns an array of table entries.

RETURNS:

Zero if unsuccessful, non-zero otherwise.

DESCRIPTION:

This function returns the values associated with a range of contiguous table indices, starting at the specified entry index. If the specified lookup table entry is not defined, the default entry for that type of table is returned.

The type of structure returned depends on the specified type.

If the requested value type is `PEXSetValue`, the values returned will be those originally set in the table entry. If the requested value type is `PEXRealizedValue`, the values returned will be those actually used during rendering (i.e. the values used if the specified entry value can not be exactly matched).

PEXlib allocates memory for the returned entry. `PEXFreeTableEntries` should be called to deallocated the memory.

DATA STRUCTURES:

```
typedef XID        PEXLookupTable;
```

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXLookupTable
The specified lookup table resource identifier is invalid.

BadValue
A specified value is out of range, or an invalid bit is set in the value mask.

SEE ALSO:

`PEXExtSetTableEntries`, `PEXSetTableEntries`, `PEXFreeTableEntries`

Extended Fetch Elements

NAME:

PEXExtFetchElements - Extended Fetch Elements

SYNTAX:

Status PEXExtFetchElements(Display **display*, PEXStructure *structure*,
int *whence1*, long *offset1*, int *whence2*, long *offset2*,
int *float_format*, unsigned long **count_return*,
unsigned long **length_return*, char ***ocs_return*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
structure	The resource identifier of the structure.
whence1	A value of specifying, with *offset1*, the first limit of the range of elements to be fetched (`PEXBeginning`, `PEXCurrent`, `PEXEnd`).
offset1	The offset from *whence1* denoting the first limit of the range of elements to be fetched.
whence2	A value specifying, with *offset2*, the second limit of the range of elements to be fetched (`PEXBeginning`, `PEXCurrent`, `PEXEnd`).
offset2	The offset from whence2 denoting the second limit of the range of elements to be fetched.
float_format	The floating point format to use when formatting the output commands to be fetched (PEXIEEE_754_32, PEXDEC_F_Floating, PEXIEEE_754_64, PEXDEC_D_Floating).
count_return	Returns the number of output commands returned.
length_return	Returns the length, in bytes, of the output commands fetched.
ocs_return	Returns a pointer to protocol-formatted output commands.

RETURNS:

Zero if unsuccessful, non-zero otherwise.

DESCRIPTION:

This function fetches a range of structure elements from the specified structure. If either computed offset is less than zero, it is set to zero before fetching the structure elements. If either computed offset is greater than the number of elements in the structure, it is set to the offset of the last structure element in the structure. The element pointer attribute of structure is not affected by this command. No information will be returned for inquiries on element offset zero.

A null pointer is returned if the requested floating point format is not supported.

Any text or annotation text output commands returned will be formatted as encoded text or encoded annotation text.

PEXlib allocates memory for the return value. `XFree` should be called to deallocate the memory.

DATA STRUCTURES:

```
typedef XID       PEXStructure;
```

ERRORS:

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXStructure
The specified structure resource identifier is invalid.

BadValue
A specified value for whence parameter is invalid.

SEE ALSO:

`PEXCreateStructure`, `PEXExtDecodeOCs`, `PEXExtEncodeOCs`, `PEXSendOCs`

Extended Query Color Approximation

NAME:

PEXExtQueryColorApprox - Find a Supported Color Approximation for the Specified Example Drawable

SYNTAX:

int PEXExtQueryColorApprox(Display **display*, Drawable *drawable*,
PEXColorApproxEntry **color_approx*, unsigned long **count_return*,
PEXColorApproxEntry ***color_approx_return*, int **all_return*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
drawable	The drawable indicates the depth, root, and visual (if the drawable has one) the application is interested in.
color_approx	A pointer to the desired color approximation. The application wants to determine if the desired color approximation is supported by windows of the same root and depth as the drawable.
count_return	Returns the number of returned alternative supported color approximations. Not valid if the function returns True.
color_approx_return	Returns a list of alternative supported color approximations. Not valid if the function returns True.
all_return	Returns True if the list of returned color approximations is the complete list of supported color approximations. Not valid if the function returns True.

RETURNS:

True if the given color approximation is supported; False otherwise. If an error condition was not identified and False is returned, the output parameters of the function provide alternative supported color approximation(s).

DESCRIPTION:

This function is used to find a supported color approximation for the given example drawable. If the given color approximation is supported by drawables of the same visual (if the drawable has one), root, and depth as the given example drawable, the function returns True.

If the given color approximation is not supported, the function returns False. In this case, a list of color approximations which are supported by drawables of the same visual (if the drawable has one), root, and depth as the given example drawable are returned. It is indicated whether this list is exhaustive.

PEXlib allocates memory if any alternative supported color approximations are supported. If PEXlib is unable to allocate the memory it requires, parameter *count_return* will be returned set to zero and parameter *color_approx_return* will be returned set to NULL. `XFree` should be called to deallocate the memory if it was allocated.

DATA STRUCTURES:

See the existing PEXlib 5.1 data structure definitions.

ERRORS:

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXDrawable
The specified drawable resource identifier is invalid.

BadMatch
The specified lookup table resource was not created with a drawable compatible with the drawable used to create the renderer resource.

BadValue
A specified value is out of range.

SEE ALSO:

`PEXSetTableEntries`, `PEXExtSetTableEntries`

▼ Extended Set Texture Mapping Perspective Correction

NAME:

PEXExtSetTMPerspectiveCorrection - Set Texture Mapping Perspective Correction

SYNTAX:

void PEXExtSetTMPerspectiveCorrection(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *method*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
method	Type of texture mapping perspective correction to apply (`PEXExtTMPerspCorrectNone`, `PEXExtTMPerspCorrectVertex`, `PEXExtTMPerspCorrectPixel`).

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *tm_perspective_correction* attribute. If the specified type is not supported by the PEX server, texture mapping perspective correction `PEXExtTMPerspCorrectNone` is used. The default value is `PEXExtTMPerspCorrectNone`.

Supported values for texture map perspective correction are inquirable using `PEXGetEnumTypeInfo`.The default texture map perspective correction is `PEXExtTMPerspCorrectNone`.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXExtSetTMSampleFrequency`

Extended Set Texture Mapping Sample Frequency

NAME:

PEXExtSetTMSampleFrequency - Set Texture Mapping Sample Frequency

SYNTAX:

void PEXExtSetTMSampleFrequency(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *frequency*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
frequency	The texture mapping sample frequency (`PEXExtTMSampleFrequencyPixel`, `PEXExtTMSampleFrequencyInterpDep`).

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *tm_sample_frequency* attribute. If the specified type is not supported by the PEX server, texture mapping sample frequency `PEXExtTMSampleFrequencyPixel` is used.

Supported values for texture map sample frequency are inquirable using `PEXGetEnumTypeInfo`.The default texture map sample frequency is `PEXExtTMSampleFrequencyPixel`.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXExtSetTMPerspectiveCorrection`

Extended Set Texture Mapping Resource Hints

NAME:

PEXExtSetTMResourceHints - Set Texture Mapping Resource Hints

SYNTAX:

void PEXExtSetTMResourceHints(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *optimization_hint*, unsigned int *count*,
PEXTableIndex **priorities*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
optimization_hint	Resource optimization approach to consider for all subsequently activated textures (`PEXExtTMResourceHintNone`, `PEXExtTMResourceHintSpeed`, `PEXExtTMResourceHintSpace`).
count	The number of indices.
priorities	An array of texture binding table indices.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *tm_resource_hints* attributes. The list of prioritized textures replaces the previous list. If a referenced TM Binding LUT entry is not defined, it is not used in resource optimization by the server.

In the pipeline context, the default *tm_resource_hints* optimization is None and the default *tm_resource_hints* prioritized textures is the empty list.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

▼ Extended Set Current Active Texture Maps

NAME:

PEXExtSetActiveTextures - Set Current Active Texture Maps

SYNTAX:

void PEXExtSetActiveTextures(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, unsigned int *count*,
PEXTableIndex **textures*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
count	Number of textures to activate.
textures	An array of texture binding table indices corresponding to the texture maps to activate.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *active_textures* attribute. This list of textures replaces the previous list. If a referenced TM Binding LUT entry is not defined, the default entry will be used instead. If the default entry is not defined, the default values for the TM Binding LUT will be applied (this results to a black and white checkerboard texture being applied). The default pipeline context value is the empty list.

The textures in the active texture list are sequentially applied in the order provided using the appropriate texture map parameterization, sampling, and composition methods specified in the TM Binding LUT. If the number of textures to be activated exceeds `PEXExtIDMaxTextureMaps`, the list of textures in beyond the first '`PEXExtIDMaxTextureMaps`' textures will be ignored.

Setting active textures with a count of zero means there are no active textures, resulting in the default texture being applied.

A texture is activated 1) if it is not already activated, and 2) when the `PEXExtOCActive-Textures` or `PEXExtOCBFActiveTextures` OC is processed with the texture in the activation list. A texture will be deactivated when it is no longer in the *active_textures* or *bf_active_textures* list.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetBFActiveTextures`

▼ Extended Set Current Back-Face Active Texture Maps

NAME:

PEXExtSetBFActiveTextures - Set Current Back-Face Active Texture Maps

SYNTAX:

void PEXExtSetBFActiveTextures(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, unsigned int *count*,
PEXTableIndex **textures*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
count	Number of textures to activate.
textures	An array of texture binding table indices corresponding to the texture maps to activate.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *bf_active_textures* attribute. This list of textures replaces the previous list. If a referenced TM Binding LUT entry is not defined, the default entry will be used instead. If the default entry is not defined, the default values for the TM Binding LUT will be applied (this results to a black and white checkerboard texture being applied). The default pipeline context value is the empty list.

The textures in the active texture list are sequentially applied in the order provided using the appropriate texture map parameterization, sampling, and composition methods specified in the TM Binding LUT. If the number of textures to be activated exceeds `PEXExtIDMaxTextureMaps`, the list of textures in beyond the first '`PEXExtIDMaxTextureMaps`' textures will be ignored.

Setting active textures with a count of zero means there are no active textures, resulting in the default texture being applied.

A texture is activated 1) if it is not already activated, and 2) when the `PEXExtOCActive-Textures` or `PEXExtOCBFActiveTextures` OC is processed with the texture in the activation list. A texture will be deactivated when it is no longer in the *active_textures* or *bf_active_textures* list.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetActiveTextures`

Extended Fill Area Set With Data

NAME:

PEXExtFillAreaSetWithData - 3D Set of Fill Areas Primitive With Additional Data

SYNTAX:

void PEXExtFillAreaSetWithData(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *shape_hint*, int *ignore_edges*,
int *contour_hint*, unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*, unsigned int *vertex_attributes*,
int *color_type*, unsigned int *count*, PEXFacetData **facet_data*,
PEXExtListOfVertex **vertex_lists*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
shape_hint	The shape which describes all of the contours (`PEXShapeComplex`, `PEXShapeNonConvex`, `PEXShapeConvex`, `PEXShapeUnknown`).
ignore_edges	A flag that determines if surface edges are rendered (True or False).
contour_hint	A flag that indicates whether contours are disjoint or overlapping (`PEXContourDisjoint`, `PEXContourNested`, `PEXContourIntersecting`, `PEXContourUnknown`).
vertex_fp_data_size	The number of floating point data values defined with each vertex.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXGAEdges`, `PEXExtGAData`).

color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
count	The number of fill areas in the set.
facet_data	A pointer to the facet data.
vertex_lists	A pointer to the list of vertex arrays defining each contour of the fill area set.

RETURNS:

None

DESCRIPTION:

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 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 in which they appear above.

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

color	COLOR
normal	VECTOR_3D
edges	SWITCH
fp_data	LISTofFLOAT

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 in which they appear in the list above.

If color values are passed as part of *per_facet_data* or as part of the *per_vert_data* 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. 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.

If normals are passed per vertex, they are taken to be the normals at the vertices of the fill area. All normals are assumed to be unit vectors. The effect of rendering with vertex normals which are not unit vectors is implementation-dependent.

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.

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.

If *vertex_fp_data_size* is non-zero, you will need to pack the vertex data yourself. The coordinate data is packed first, followed by the color, normal, and edge data (if present), in exactly the same order as defined by data structure definitions `PEXCoord` through `PEXVertexRGB16NormalEdge`. The floating point values directly follow.

For example, if an application has the coordinates, normal, and three floating point data list values for a vertex, it would pack the data as follows:

```
typedef struct {
    /*  Use an existing PEXArrayOfVertex member and add the floating
      *  point data list:  */
    PEXVertexNormalcoords_and_normal;
    floatfp_data[3];
} MyPEXVertexNormalFPData3;

MyPEXVertexNormalFPData3 my_vertex_data;
PEXExtArrayOfVertex vertex_data;

my_vertex_data.coords_and_normal.point.x = MY_VERTEX_X;
my_vertex_data.coords_and_normal.point.y = MY_VERTEX_Y;
my_vertex_data.coords_and_normal.point.z = MY_VERTEX_Z;
my_vertex_data.coords_and_normal.normal.x = MY_VERTEX_NORMAL_X;
```

```
my_vertex_data.coords_and_normal.normal.y = MY_VERTEX_NORMAL_Y;
my_vertex_data.coords_and_normal.normal.z = MY_VERTEX_NORMAL_Z;
my_vertex_data.fp_data[0] = MY_VERTEX_FP_DATA_VALUE_1;
my_vertex_data.fp_data[1] = MY_VERTEX_FP_DATA_VALUE_2;
my_vertex_data.fp_data[2] = MY_VERTEX_FP_DATA_VALUE_3;

vertex_data.with_fp_data = (PEXPointer)(&my_vertex_data);
```

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetInteriorStyle`, `PEXSetInteriorStyleIndex`, `PEXSetSurfaceColorIndex`, `PEXSetSurfaceColor`, `PEXSetReflectionAttributes`, `PEXSetReflectionModel`, `PEXSetSurfaceInterpMethod`, `PEXSetBFInteriorStyle`, `PEXSetBFInteriorStyleIndex`, `PEXSetBFSurfaceColorIndex`, `PEXSetBFSurfaceColor`, `PEXSetBFReflectionAttributes`, `PEXSetBFReflectionModel`, `PEXSetBFSurfaceInterpMethod`, `PEXSetFacetCullingMode`, `PEXSetFacetDistinguishFlag`, `PEXSetPatternSize`, `PEXSetPatternAttributes`, `PEXSetPatternAttributes2D`, `PEXSetInteriorBundleIndex`, `PEXSetSurfaceEdgeFlag`, `PEXSetSurfaceEdgeType`, `PEXSetSurfaceEdgeWidth`, `PEXSetSurfaceEdgeColor`, `PEXSetSurfaceEdgeColorIndex`, `PEXSetEdgeBundleIndex`

Extended Set of Fill Area Sets

NAME:

PEXExtSetOfFillAreaSets - 3D Set of Fill Area Sets Primitive

SYNTAX:

void PEXExtSetOfFillAreaSets(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *shape_hint*,
unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*, unsigned int *vertex_attributes*,
unsigned int *edge_attributes*, int *contour_hint*,
int *contours_all_one*, int *color_type*, unsigned int *set_count*,
PEXArrayOfFacetData *facet_data*, unsigned int *vertex_count*,
PEXExtArrayOfVertex *vertices*, unsigned int *index_count*,
PEXSwitch **edge_flags*, PEXConnectivityData **connectivity*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
shape_hint	The shape which describes all of the contours (`PEXShapeComplex`, `PEXShapeNonConvex`, `PEXShapeConvex`, `PEXShapeUnknown`).
vertex_fp_data_size	The number of floating point data values defined per vertex fill area set attributes.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXExtGAData`).
edge_attributes	A mask indicating the edge attributes provided (`PEXGANone`, or `PEXGAEdges`).

contour_hint	A flag that indicates whether contours are disjoint or overlapping (`PEXContourDisjoint`, `PEXContourNested`, `PEXContourIntersecting`, `PEXContourUnknown`)
contours_all_one	True if each fill area set contains only one contour; False otherwise.
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
set_count	The number of fill area sets.
facet_data	An array of facet data.
vertex_count	The number of vertices.
vertices	An array of vertices.
index_count	The number of vertex connectivity indices (also number of edge flags, if edges are specified).
edge_flags	An array of edge flags.
connectivity	A pointer to the list of contour connectivity data.

RETURNS:

None

DESCRIPTION:

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 nothing is known about the shape of the constituent contours. A shape of Complex means some contours of the fill area sets may have edges that self-intersect. A shape of Nonconvex means none of the contours of the fill area sets have edges that self-intersect, but some may not be wholly convex. Convex means all of the interior angles of all of the contours are convex. (Note a fill area set with more than one contour is always allowed to have contours that intersect. It is quite possible the only times 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 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 relationship 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 which 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 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 which are passed in will be of the type specified by *color_type*.

The parameter *fas_attributes* indicates the attributes which 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 passed in this way must be passed in the order in which they appear in the list above. 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 which are provided with 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
fp_data	LISTofFLOAT

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 which are passed in this way must be passed in the order in which they appear in the list above. There will always be *num_verts* entries in the *per_vert_data* list.

If color values are passed as part of *per_facet_data* or as part of the *per_vert_data* 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. 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.

All normals are assumed to be unit vectors. The effect of rendering with facet or vertex normals which are not unit vectors is implementation dependent.

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 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 recoreds, one for each of the contours in the fill rea 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 fist contour. Depending on n, the next value in the list is either the number of vertices in the second contour for the first fill ara 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 which 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.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetInteriorStyle`, `PEXSetInteriorStyleIndex`, `PEXSetSurfaceColorIndex`, `PEXSetSurfaceColor`, `PEXSetReflectionAttributes`, `PEXSetReflectionModel`, `PEXSetSurfaceInterpMethod`, `PEXSetBFInteriorStyle`, `PEXSetBFInteriorStyleIndex`, `PEXSetBFSurfaceColorIndex`, `PEXSetBFSurfaceColor`, `PEXSetBFReflectionAttributes`, `PEXSetBFReflectionModel`, `PEXSetBFSurfaceInterpMethod`, `PEXSetFacetCullingMode`, `PEXSetFacetDistinguishFlag`, `PEXSetPatternSize`, `PEXSetPatternAttributes`, `PEXSetPatternAttributes2D`, `PEXSetInteriorBundleIndex`, `PEXSetSurfaceEdgeFlag`, `PEXSetSurfaceEdgeType`, `PEXSetSurfaceEdgeWidth`, `PEXSetSurfaceEdgeColor`, `PEXSetSurfaceEdgeColorIndex`, `PEXSetEdgeBundleIndex`

▼ Extended Triangle Strip

NAME:

PEXExtTriangleStrip - 3D Triangle Strip Primitive

SYNTAX:

void PEXExtTriangleStrip(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*, unsigned int *vertex_attributes*,
int *color_type*, PEXArrayOfFacetData *facet_data*,
unsigned int *count*, PEXExtArrayOfVertex *vertices*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
vertex_fp_data_size	The number of floating point data values defined per vertex.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXExtGAData`).
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
facet_data	An array of facet data.
count	The number of vertices.
vertices	An array of vertices defining the triangle strip.

RETURNS:

None

DESCRIPTION:

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 in which they appear in the list above. 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
fp_data	LISTofFLOAT

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 which are passed in this way must be passed in the order in which they appear in the list above.

If color values are passed as part of *per_facet_data* or as part of the *per_vert_data* 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. 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.

All normals are assumed to be unit vectors. The effect of rendering with vertex normals which are not unit vectors is implementation-dependent.

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 which 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.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetInteriorStyle`, `PEXSetInteriorStyleIndex`, `PEXSetSurfaceColorIndex`, `PEXSetSurfaceColor`, `PEXSetReflectionAttributes`, `PEXSetReflectionModel`, `PEXSetSurfaceInterpMethod`, `PEXSetBFInteriorStyle`, `PEXSetBFInteriorStyleIndex`, `PEXSetBFSurfaceColorIndex`, `PEXSetBFSurfaceColor`, `PEXSetBFReflectionAttributes`, `PEXSetBFReflectionModel`, `PEXSetBFSurfaceInterpMethod`, `PEXSetFacetCullingMode`, `PEXSetFacetDistinguishFlag`, `PEXSetPatternSize`, `PEXSetPatternAttributes`, `PEXSetPatternAttributes2D`, `PEXSetInteriorBundleIndex`, `PEXSetSurfaceEdgeFlag`, `PEXSetSurfaceEdgeType`, `PEXSetSurfaceEdgeWidth`, `PEXSetSurfaceEdgeColor`, `PEXSetSurfaceEdgeColorIndex`, `PEXSetEdgeBundleIndex`

Extended Quadrilateral Mesh

NAME:

PEXExtQuadrilateralMesh - 3D Quadrilateral Mesh Primitive

SYNTAX:

void PEXExtQuadrilateralMesh(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *shape_hint*,
unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*, unsigned int *vertex_attributes*,
int *color_type*, PEXArrayOfFacetData *facet_data*,
unsigned int *col_count*, unsigned int *row_count*,
PEXExtArrayOfVertex *vertices*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
shape_hint	The shape which describes all of the contours (`PEXShapeComplex`, `PEXShapeNonConvex`, `PEXShapeConvex`, `PEXShapeUnknown`).
vertex_fp_data_size	The number of floating point data values defined per vertex.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXExtGAData`).
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
facet_data	An array of facet data.
col_count	The number of columns in the vertex array.

row_count The number of rows in the vertex array.

vertices A two-dimensional (row-major) array of vertices defining the quadrilateral mesh.

RETURNS:

None

DESCRIPTION:

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 which comprise the primitive. A shape hint of Unknown means nothing is known about the shape of the constituent quadrilaterals. A shape of Complex means some of the quadrilaterals may have edges which self-intersect. A shape of Nonconvex means none of the quadrilaterals have edges which self-intersect, but some may not be wholly convex. Convex means all of the interior angles of all of the quadrilaterals are convex. Quadrilaterals of a higher complexity than indicated by their shape hint are drawn in an implementation-dependent manner. Color values which 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 which are passed in this way must be passed in the order in which they appear in the list above.

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
fp_data	LISTofFLOAT

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 which are passed in this way must be passed in the order in which they appear in the list above.

If color values are passed as part of *per_facet_data* or as part of the *per_vert_data* 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. 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.

All normals are assumed to be unit vectors. The effect of rendering with vertex normals which are not unit vectors is implementation-dependent.

The surface will be created from a vertex array which 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* x *n_pts* entries in the vertices array, and there are (m_pts-1) x (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 which affect the representation of fill area sets also affect the representation of 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.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetInteriorStyle`, `PEXSetInteriorStyleIndex`, `PEXSetSurfaceColorIndex`, `PEXSetSurfaceColor`, `PEXSetReflectionAttributes`, `PEXSetReflectionModel`, `PEXSetSurfaceInterpMethod`, `PEXSetBFInteriorStyle`, `PEXSetBFInteriorStyleIndex`, `PEXSetBFSurfaceColorIndex`, `PEXSetBFSurfaceColor`, `PEXSetBFReflectionAttributes`, `PEXSetBFReflectionModel`, `PEXSetBFSurfaceInterpMethod`, `PEXSetFacetCullingMode`, `PEXSetFacetDistinguishFlag`, `PEXSetPatternSize`, `PEXSetPatternAttributes`, `PEXSetPatternAttributes2D`, `PEXSetInteriorBundleIndex`, `PEXSetSurfaceEdgeFlag`, `PEXSetSurfaceEdgeType`, `PEXSetSurfaceEdgeWidth`, `PEXSetSurfaceEdgeColor`, `PEXSetSurfaceEdgeColorIndex`, `PEXSetEdgeBundleIndex`

▼ Extended Set Primitive Anti-Aliasing

NAME:

PEXExtSetPrimitiveAA - Set Primitive Anti-Aliasing

SYNTAX:

void PEXExtSetPrimitiveAA(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *mode*, int *blend_op*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
mode	The primitive anti-aliasing mode (`PEXExtPrimAANone`, `PEXExtPrimAAPoint`, `PEXExtPrimAAVector`, `PEXExtPrimAAPointVector`, `PEXExtPrimAAPolygon`, `PEXExtPrimAAPointPolygon`, `PEXExtPrimAAVectorPolygon`, or `PEXExtPrimAAPointVectorPolygon`).
blend_op	The primitive anti-aliasing blend operation (`PEXExtPrimAABlendOpImpDep` or `PEXExtPrimAABlendOpSimpleAlpha`).

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *primitive_aa* mode and *primitive_aa blend_op* attributes. If the specified mode is not supported by the server, primitive anti-aliasing mode `PEXExtPrimAANone` is used. If the specified blend operation is not supported by the server, primitive anti-aliasing blend `PEXExtPrimAABlendOpImpDep` is used. In the default pipeline context, the primitive anti-aliasing mode is `PEXExtPrimAANone` and the primitive anti-aliasing blend operation is `PEXExtPrimAABlendOpImpDep`.

Any integer may be specified as the primitive anti-aliasing mode or the primitive anti-aliasing blend operation in this output command. If an inconsistent value is specified in the data record, an `OutputCommand` error is reported.

Primitive anti-aliasing interacts with `HLHSR` in an implementation-dependent manner.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXGetEnumTypeInfo`

▼ Extended Set Line Cap Style

NAME:

PEXExtSetLineCapStyle - Set Line Cap Style

SYNTAX:

void PEXExtSetLineCapStyle(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *style*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
style	The cap style to be applied to wide lines (`PEXExtLineCapStyleButt`, `PEXExtLineCapStyleRound`, or `PEXExtLineCapStyleProjecting`).

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *line_cap_style* attribute. If the specified line cap style is not supported by the server, line cap style `PEXExtLineCapStyleButt` is used. In the default pipeline context, the line cap style is `PEXExtLineCapStyleButt`.

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

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXGetEnumTypeInfo`

▼ Extended Set Line Join Style

NAME:

PEXExtSetLineJoinStyle - Set Line Join Style

SYNTAX:

void PEXExtSetLineJoinStyle(Display **display*, XID *resource_id*,
PEXOCRequestType *req_type*, int *style*)

PARAMETERS:

display — A pointer to a display structure returned by a successful `XOpenDisplay` call.

resource_id — The resource identifier of the renderer or structure.

req_type — The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).

style — The join style to be applied to wide lines (PEXExtLineJoinStyleImpDep, PEXExtLineJoinStyleRound, PEXExtLineJoinStyleMiter, or PEXExtLineJoinStyleBevel).

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will modify the renderer's *line_join_style* attribute. If the specified line join style is not supported by the server, line join style `PEXExtLineJoinStyleImpDep` is used. In the default pipeline context, the line cap style is `PEXExtLineJoinStyleImpDep`.

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

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXGetEnumTypeInfo`

▼ Extended 3D Ellipse

NAME:

PEXExtEllipse - 3D Ellipse

SYNTAX:

void PEXExtEllipse(Display **display*, XID *resource_id*, PEXOCRequest *req_type*,
PEXCoord **center*, PEXVector **major*, PEXVector **minor*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
center	The center of the ellipse in modelling coordinates.
major	The major axis of the ellipse in modelling coordinates.
minor	The minor axis of the ellipse in modelling coordinates.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will cause an ellipse to be rendered. The ellipse's center and axes are defined in three dimensional modelling coordinate space. The major axis defines the positive X-axis of the ellipse local coordinate system. The minor axis is used in defining the positive Y-axis of the ellipse local coordinate system.

The major and minor axis parameters define a rectangle which circumscribes the ellipse and is anchored at the center of the ellipse. Two sides are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to midpoints of sides of the rectangle. The ellipse is tangent to the sides of the rectangle at their midpoints. The behavior if the major and minor axes are not orthogonal is device-dependent.

The current values of the fill area attributes are applied to the primitive. The ellipse is rendered according to the curve approximation criteria.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetInteriorStyle`, `PEXSetInteriorStyleIndex`, `PEXSetSurfaceColorIndex`, `PEXSetSurfaceColor`, `PEXSetReflectionAttributes`, `PEXSetReflectionModel`, `PEXSetSurfaceInterpMethod`, `PEXSetBFInteriorStyle`, `PEXSetBFInteriorStyleIndex`, `PEXSetBFSurfaceColorIndex`, `PEXSetBFSurfaceColor`, `PEXSetBFReflectionAttributes`, `PEXSetBFReflectionModel`, `PEXSetBFSurfaceInterpMethod`, `PEXSetFacetCullingMode`, `PEXSetFacetDistinguishFlag`, `PEXSetPatternSize`, `PEXSetPatternAttributes`, `PEXSetPatternAttributes2D`, `PEXSetInteriorBundleIndex`, `PEXSetSurfaceEdgeFlag`, `PEXSetSurfaceEdgeType`, `PEXSetSurfaceEdgeWidth`, `PEXSetSurfaceEdgeColor`, `PEXSetSurfaceEdgeColorIndex`, `PEXSetEdgeBundleIndex`, `PEXSetCurveApprox`

▼ Extended 2D Ellipse

NAME:

PEXExtEllipse2D - 2D Ellipse

SYNTAX:

void PEXExtEllipse2D(Display **display*, XID *resource_id*, PEXOCRequest *req_type*,
PEXCoord2D **center*, PEXVector2D **major*, PEXVector2D **minor*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
center	The center of the ellipse in 2D modelling coordinates.
major	The major axis of the ellipse in 2D modelling coordinates.
minor	The minor axis of the ellipse in 2D modelling coordinates.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will cause an ellipse to be rendered. The ellipse's center and axes are defined in two dimensional modelling coordinate space. The major axis defines the positive X-axis of the ellipse local coordinate system. The minor axis is used in defining the positive Y-axis of the ellipse local coordinate system. The ellipse, centered about the center coordinates, will lie in the modelling coordinate XY plane (Z = 0).

The major and minor axis parameters define a rectangle which circumscribes the ellipse and is anchored at the center of the ellipse. Two sides are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to midpoints of sides of the rectangle. The ellipse is tangent to the sides of the rectangle at their midpoints. The behavior if the major and minor axes are not orthogonal is device-dependent.

The current values of the fill area attributes are applied to the primitive. The ellipse is rendered according to the curve approximation criteria.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetInteriorStyle`, `PEXSetInteriorStyleIndex`, `PEXSetSurfaceColorIndex`, `PEXSetSurfaceColor`, `PEXSetReflectionAttributes`, `PEXSetReflectionModel`, `PEXSetSurfaceInterpMethod`, `PEXSetBFInteriorStyle`, `PEXSetBFInteriorStyleIndex`, `PEXSetBFSurfaceColorIndex`, `PEXSetBFSurfaceColor`, `PEXSetBFReflectionAttributes`, `PEXSetBFReflectionModel`, `PEXSetBFSurfaceInterpMethod`, `PEXSetFacetCullingMode`, `PEXSetFacetDistinguishFlag`, `PEXSetPatternSize`, `PEXSetPatternAttributes`, `PEXSetPatternAttributes2D`, `PEXSetInteriorBundleIndex`, `PEXSetSurfaceEdgeFlag`, `PEXSetSurfaceEdgeType`, `PEXSetSurfaceEdgeWidth`, `PEXSetSurfaceEdgeColor`, `PEXSetSurfaceEdgeColorIndex`, `PEXSetEdgeBundleIndex`, `PEXSetCurveApprox`

Extended 2D Circle

NAME:

PEXExtCircle2D - 2D Circle

SYNTAX:

void PEXExtCircle2D(Display **display*, XID *resource_id*,
PEXOCRequest *req_type*, PEXCoord2D **center*, double *radius*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
center	The center of the circle in 2D modelling coordinates.
radius	The radius of the circle in modelling coordinates.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will cause a circle to be rendered. The circle's center and radius are defined in two dimensional modelling coordinate space. A circle having the specified radius is drawn centered about the center coordinates in the modelling coordinate Z=0 plane.

The current values of the fill area attributes are applied to the primitive. The circle is rendered according to the curve approximation criteria.

If the radius of the circle is less than or equal to zero, a dot at the center will be displayed.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

PEXSetInteriorStyle, PEXSetInteriorStyleIndex, PEXSetSurfaceColorIndex, PEXSetSurfaceColor, PEXSetReflectionAttributes, PEXSetReflectionModel, PEXSetSurfaceInterpMethod, PEXSetBFInteriorStyle, PEXSetBFInteriorStyleIndex, PEXSetBFSurfaceColorIndex, PEXSetBFSurfaceColor, PEXSetBFReflectionAttributes, PEXSetBFReflectionModel, PEXSetBFSurfaceInterpMethod, PEXSetFacetCullingMode, PEXSetFacetDistinguishFlag, PEXSetPatternSize, PEXSetPatternAttributes, PEXSetPatternAttributes2D, PEXSetInteriorBundleIndex, PEXSetSurfaceEdgeFlag, PEXSetSurfaceEdgeType, PEXSetSurfaceEdgeWidth, PEXSetSurfaceEdgeColor, PEXSetSurfaceEdgeColorIndex, PEXSetEdgeBundleIndex, PEXSetCurveApprox

▼ Extended 3D Elliptical Arc

NAME:

PEXExtEllipticalArc - 3D Elliptical Arc

SYNTAX:

void PEXExtEllipticalArc(Display **display*, XID *resource_id*,
PEXOCRequest *req_type*, PEXCoord **center*, PEXVector **major*,
PEXVector **minor*, double *start*, double *stop*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
center	The center of the elliptical arc in modelling coordinates.
major	The major axis of the elliptical arc in modelling coordinates.
minor	The minor axis of the elliptical arc in modelling coordinates.
start	The start angle in radians.
stop	The stop angle in radians.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will cause an elliptical arc to be rendered. The elliptical arc's center and axes are defined in three dimensional modelling coordinate space. The major axis defines the positive X-axis of the ellipse local coordinate system. The minor axis is used in defining the positive Y-axis of the ellipse local coordinate system. A start and end angle relative to the positive X axis define the portion of the circumference drawn to form the specified arc.

The arc is drawn in a counter clockwise direction from the starting angle position to the ending angle position.

The major and minor axis parameters define a rectangle which circumscribes the ellipse and is anchored at the center of the ellipse. Two sides are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to midpoints of sides of the parallelogram. The ellipse is tangent to the sides of the parallelogram at their midpoints. The current values of the line attributes are applied to the primitive. The elliptical arc is rendered according to the curve approximation criteria.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetLineType`, `PEXSetLineWidth`, `PEXSetLineColorIndex`, `PEXSetLineColor`, `PEXSetPolylineInterpMethod`, `PEXSetLineBundleIndex`, `PEXSetCurveApprox`

▼ Extended 2D Elliptical Arc

NAME:

PEXExtEllipticalArc2D - 2D Elliptical Arc

SYNTAX:

void PEXExtEllipticalArc2D(Display **display*, XID *resource_id*,
PEXOCRequest *req_type*, PEXCoord2D **center*, PEXVector2D **major*,
PEXVector2D **minor*, double *start*, double *stop*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
center	The center of the elliptical arc in2D modelling coordinates.
major	The major axis of the elliptical in 2D modelling coordinates.
minor	The minor axis of the elliptical in 2D modelling coordinates.
start	The start angle in radians.
stop	The stop angle in radians.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will cause an elliptical arc to be rendered. The elliptical arc's center and axes are defined in two dimensional modelling coordinate space. The major axis defines the positive X-axis of the ellipse local coordinate system. The minor axis is used in defining the positive Y-axis of the ellipse local coordinate system. A start and end angle relative to the positive X axis define the portion of the circumference drawn to form the specified arc. The arc will lie in the modelling coordinate XY plane (Z = 0).

The arc is drawn in a counter clockwise direction from the starting angle position to the ending angle position in the Z=0 plane.

The major and minor axis parameters define a rectangle which circumscribes the ellipse and is anchored at the center of the ellipse. Two sides are parallel to the major axis. The other two sides are parallel to the minor axis. The major and minor axes extend from the center of the ellipse to midpoints of sides of the parallelogram. The ellipse is tangent to the sides of the parallelogram at their midpoints. The current values of the line attributes are applied to the primitive. The elliptical arc is rendered according to the curve approximation criteria.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetLineType`, `PEXSetLineWidth`, `PEXSetLineColorIndex`, `PEXSetLineColor`, `PEXSetPolylineInterpMethod`, `PEXSetLineBundleIndex`, `PEXSetCurveApprox`

Extended 2D Circular Arc

NAME:

PEXExtCircularArc2D - 2D Circular Arc

SYNTAX:

void PEXExtCircularArc2D(Display **display*, XID *resource_id*,
PEXOCRequest *req_type*, PEXCoord2D **center*, double *radius*,
double *start*, double *stop*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output command (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).
center	The center of the circular arc in2D modelling coordinates.
radius	The radius of the circular arc in 2D modelling coordinates.
start	The start angle in radians.
stop	The stop angle in radians.

RETURNS:

None

DESCRIPTION:

When processed by a renderer, this command will cause a circular arc to be rendered. The circular arc's center and radius are defined in two dimensional modelling coordinate space. A center point and a radius specify the circle from which the arc is drawn. A start and end angle relative to the positive X axis define the portion of the circumference drawn to form the specified arc. The arc is drawn in a counter clockwise direction from the starting angle position to the ending angle position in the Z=0 plane.

The current values of the line attributes are applied to the primitive. The circular arc is rendered according to the curve approximation criteria.

If the radius of the circular arc is less than or equal to zero, a dot at the center will be displayed.

ERRORS:

BadPEXRenderer
The specified renderer resource identifier is invalid.

BadPEXStructure
The specified structure resource identifier is invalid.

SEE ALSO:

`PEXSetLineType`, `PEXSetLineWidth`, `PEXSetLineColorIndex`, `PEXSetLineColor`, `PEXSetPolylineInterpMethod`, `PEXSetLineBundleIndex`, `PEXSetCurveApprox`

▼ Extended Set All Pipeline Context Attributes Value Mask

NAME:

PEXExtSetPCAttributeMaskAll - Macro to Set All Pipeline Context Attributes Value Mask

SYNTAX:

PEXExtSetPCAttributeMaskAll(*mask*)

PARAMETERS:

mask The address of the value mask - an array of three unsigned long.

DESCRIPTION:

This is a utility macro to aid in setting up the bitmask for the extended and the non-extended pipeline context attributes. This macro will set all valid bits in the mask.

ERRORS:

None

SEE ALSO:

`PEXCreatePipelineContext`, `PEXExtChangePipelineContext`, `PEXCopyPipelineContext`, `PEXExtGetPipelineContext`, `PEXExtSetPCAttributeMask`

Extended Set Pipeline Context Attributes Value Mask

NAME:

PEXExtSetPCAttributeMask - Macro to Setup Extended Pipeline Context Attributes Value Mask

SYNTAX:

PEXExtSetPCAttributeMask(*mask*, *ext_attr*)

PARAMETERS:

mask	The address of the value mask - an array of three unsigned long.
ext_attr	A single pipeline constant attribute bitmask constant or extended pipeline constant attribute bitmask constant.

DESCRIPTION:

This is a utility macro to aid in setting up the bitmask for the extended and the non-extended pipeline context attributes.

The following attribute bitmask constants must be used:

```
PEXPCMarkerType                 <the 5.1 constants>
:
:
PEXPCParaSurfCharacteristics
PEXExtPCTMPerspectiveCorrection   <the extended constants>
PEXExtPCTMResourceHints
PEXExtPCTMSampleFrequency
PEXExtPCActiveTextures
PEXExtPCBFActiveTextures
PEXExtPCPrimitiveAA
PEXExtPCLineCapStyle
PEXExtPCLineJoinStyle
```

Note this macro does multiple evaluations of the value for *ext_attr*.

ERRORS:

None

SEE ALSO:

```
PEXCreatePipelineContext, PEXExtChangePipelineContext,
PEXCopyPipelineContext, PEXExtGetPipelineContext,
PEXExtSetPCAttributeMaskAll
```

Extended Free Pipeline Context Attributes

NAME:

PEXExtFreePCAttributes - Free Storage Returned by `PEXExtGetPipelineContext`

SYNTAX:

void PEXExtFreePCAttributes(PEXExtPCAttributes **values*)

PARAMETERS:

values A pointer to the extended pipeline context attribute values.

RETURNS:

None

DESCRIPTION:

This function deallocates memory returned by `PEXExtGetPipelineContext`.

ERRORS:

None

SEE ALSO:

`PEXExtGetPipelineContext`

Extended Set All Renderer Attribute Value Mask

NAME:

PEXExtSetRendererAttributeMaskAll - Macro to Set All Extended Renderer Attributes in Value Mask

SYNTAX:

PEXExtSetRendererAttributeMaskAll(*mask*)

PARAMETERS:

mask The address of the value mask - an array of two unsigned long.

DESCRIPTION:

This is a utility macro to aid in setting up the bitmask for the extended and the non-extended renderer attributes. This macro will set all valid bits in the mask.

ERRORS:

None

SEE ALSO:

`PEXExtChangeRenderer`, `PEXExtGetRendererAttributes`

Extended Set Renderer Attributes Mask

NAME:

PEXExtSetRendererAttributeMask - Macro to Setup Extended Renderer Attributes Value Mask

SYNTAX:

PEXExtSetRendererAttributeMask(*mask*, *ext_attr*)

PARAMETERS:

mask	The address of the value mask - an array of two unsigned long.
ext_attr	A single extended renderer attribute bitmask constant.

DESCRIPTION:

This is a utility macro to aid in setting up the bitmask for the extended and the non-extended renderer attributes.

The following attribute bitmask constants must be used:

```
PEXRAPipelineContext
:
:
PEXRAEchoMode
PEXExtRATMBindingTable
PEXExtRATMCoordSourceTable
PEXExtRATMCompositionTable
PEXExtRATMSamplingTable
```

Note this macro does multiple evaluations of the value for *ext_attr*.

ERRORS:

None

SEE ALSO:

`PEXExtChangeRenderer`, `PEXExtGetRendererAttributes`

Extended Free Renderer Attributes

NAME:

PEXExtFreeRendererAttributes - Free Storage Returned by `PEXExtGetRendererAttributes`

SYNTAX:

void PEXExtFreeRendererAttributes(PEXExtRendererAttributes **values*)

PARAMETERS:

values A pointer to the extended renderer attribute values.

RETURNS:

None

DESCRIPTION:

This function deallocates memory returned by `PEXExtGetRendererAttributes`.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtGetRendererAttributes`

▼ Extended Compute Fill Area Set With Data Texture Coordinates

NAME:

PEXExtTMCoordFillAreaSetWithData - computes a projection of texture coordinates for a fill area set with data

SYNTAX:

int PEXExtTMCoordFillAreaSetWithData(PEXExtTMCoordData **tm_coord_data*,
unsigned int *vertex_fp_data_size*, unsigned int *facet_attributes*,
unsigned int *vertex_attributes*, int *color_type*,
unsigned int *count*, PEXFacetData **facet_data*,
PEXExtListOfVertex **vertex_lists*)

PARAMETERS:

tm_coord_data	A pointer to parameterization data for this primitive (See *Description* below)
vertex_fp_data_size	The number of floating point data values defined with each vertex.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXGAEdges`, `PEXExtGAData`).
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
count	The number of fill areas in the set.
facet_data	A pointer to the facet data.
vertex_lists	A pointer to the list of vertex arrays defining each contour of the fill area set.

RETURNS:

Zero if successful; otherwise, one of the following:

`PEXExtBadPrimitive`	Normals are not valid.
`PEXExtBadValue`	Projection type is not valid.
`PEXExtBadValue`	Texture coordinate location not valid.
`PEXExtBadAlloc`	Internal allocation error has occurred.

DESCRIPTION:

This function computes a projection of texture coordinates onto a fill area set with data and stores them in the specified vertex data fields.

To compute texture coordinates, the following fields in the *tm_coord_data* structure are initialized:

projection	Texture coordinate projection to perform (`PEXExtTMProjectionSphereWC`, `PEXExtTMProjectionCylinderWC`, `PEXExtTMProjectionLinearWC`)
matrix	If projection is either `PEXExtTMProjectionSphereWC` or `PEXExtTMProjectionCylinderWC`, this transform is applied to vertices and vertex normals after the *mc_transform* has been applied. It is used to orient the data relative to the projection object (sphere, or cylinder) as described.
p0, p1	If projection is `PEXExtTMProjectionLinearWC`, then these elements contain linear projection equations. (See description below.)
coord_source	Source coordinate to compute projections specified with projection type above (`PEXExtTMCoordSourceVertexCoord` or `PEXExtTMCoordSourceVertexNormal`).
fp_data_index	Index within the vertex floating point data list in which to store the calculated texture coordinates. Space must already exist in the vertex floating point data list and the index must point to a valid location.
mc_transform	This transform is applied to vertices and vertex normals in model coordinates prior to computing the specified projection.

Additionally, for the given projections, the following information must be provided. For projections listed below, those which require a unit direction vector (s0,s1,s2) shall have one determined by the coordinate source selection previously mentioned. If `PEXExtTMCoordSourceVertexCoord` is selected, a ray is computed from the WC origin through the vertex to the interior of the projection object. If `PEXExtTMCoordSourceVertexNormal` is selected, the primitive's vertex normals (given or implicit) are used to determine the direction. The resulting values (t0,t1) are the texture coordinates stored at the requested index.

`PEXExtTMProjectionSphereWC`

The infinite sphere has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given the specified coordinate source, a direction vector (s0,s1,s2) is computed to determine a point on the interior of the sphere. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
    (-arctan(s2/s0) / (2 * PI) +
      ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
    )

t1 = arcsin(s1) / PI + 0.5

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1
```

`PEXExtTMProjectionCylinderWC`

The infinite cylinder has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given coordinate sources which are directional (normal or - reflection), a direction vector (s0,s1,s2) is computed to determine a point on the interior of the cylinder. If the coordinate source is vertex coord, a ray perpendicular to the +Y axis through the vertex is computed to determine a point and its height on the interior of the cylinder. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
    (-arctan(s2/s0) / (2 * PI) +
      ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
    )

if (coord_source == normal)
    t1 = arcsin(s1) / PI + 0.5
```

else
t1 = y_coord

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1

`PEXExtTMProjectionLinearWC`

Parameters for two linear equations are defined a p0 and p1. Given the specified coordinate source, the values will be used directly as (s0,s1,s2) to linearly project the values by the equations listed below. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list. (It should be noted that "s3" is assumed to be one as shown in the equations below.)

t0 = s0 * p0[0] + s1 * p0[1] + s2 * p0[2] + 1 * p0[3]
t1 = s0 * p1[0] + s1 * p1[1] + s2 * p1[2] + 1 * p1[3]

It should be noted that the projection of anything other than vertices for this option will lead to unpredictable visual

It is assumed the vertex list of the specified primitive has sufficient space to insert the needed texture coordinates. If space (indicated by `PEXExtGAData`) or valid indices for texture coordinates do not exist, an error will be returned and the data will remain unchanged.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`, `PEXExtTMCoordQuadrilateralMesh`

Extended Compute Set Of Fill Area Set Texture Coordinates

NAME:

PEXExtTMCoordSetOfFillAreaSets - computes a projection of texture coordinates for a set of fill are sets

SYNTAX:

int PEXExtTMCoordSetOfFillAreaSets(PEXExtTMCoordData **tm_coord_data*,
unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*, unsigned int *vertex_attributes*,
int *color_type*, unsigned int *set_count*,
PEXArrayOfFacetData *facet_data*, unsigned int *vertex_count*,
PEXExtArrayOfVertex *vertices*, unsigned int *index_count*,
PEXConnectivityData **connectivity*)

PARAMETERS:

tm_coord_data	A pointer to parameterization data for this primitive (See *Description* in `PEXExtTMCoordFillAreaSetWithData`).
vertex_fp_data_size	The number of floating point data values defined per vertex fill area set attributes.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXExtGAData`).
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
set_count	The number of fill area sets.
facet_data	An array of facet data.
vertex_count	The number of vertices.
vertices	An array of vertices.

index_count	The number of vertex connectivity indices (also number of edge flags, if edges are specified)
connectivity	A pointer to the list of contour connectivity data.

RETURNS:

Zero if successful; otherwise, one of the following:

`PEXExtBadPrimitive`	Normals are not valid.
`PEXExtBadValue`	Projection type is not valid.
`PEXExtBadValue`	Texture coordinate location not valid.
`PEXExtBadAlloc`	Internal allocation error has occurred.

DESCRIPTION:

This function computes a projection of texture coordinates onto a set of fill area sets and stores them in the specified vertex data fields.

To compute texture coordinates, the following fields in the *tm_coord_data* structure are initialized:

projection	Texture coordinate projection to perform (`PEXExtTMProjectionSphereWC`, `PEXExtTMProjectionCylinderWC`, `PEXExtTMProjectionLinearWC`)
matrix	If projection is either `PEXExtTMProjectionSphereWC` or `PEXExtTMProjectionCylinderWC`, this transform is applied to vertices and vertex normals after the *mc_transform* has been applied. It is used to orient the data relative to the projection object (sphere, or cylinder) as described.
p0, p1	If projection is `PEXExtTMProjectionLinearWC`, then these elements contain linear projection equations. (See description below.)
coord_source	Source coordinate to compute projections specified with projection type above (`PEXExtTMCoordSourceVertexCoord` or `PEXExtTMCoordSourceVertexNormal`).
fp_data_index	Index within the vertex floating point data list in which to store the calculated texture coordinates. Space must already exist in the vertex floating point data list and the index must point to a valid location.

mc_transform This transform is applied to vertices and vertex normals in model coordinates prior to computing the specified projection.

Additionally, for the given projections, the following information must be provided. For projections listed below, those which require a unit direction vector (s0,s1,s2) shall have one determined by the coordinate source selection previously mentioned. If `PEXExtTMCoordSourceVertexCoord` is selected, a ray is computed from the WC origin through the vertex to the interior of the projection object. If `PEXExtTMCoordSourceVertexNormal` is selected, the primitive's vertex normals (given or implicit) are used to determine the direction. The resulting values (t0,t1) are the texture coordinates stored at the requested index.

`PEXExtTMProjectionSphereWC`

The infinite sphere has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given the specified coordinate source, a direction vector (s0,s1,s2) is computed to determine a point on the interior of the sphere. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
     (-arctan(s2/s0) / (2 * PI) +
        ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
     )

t1 = arcsin(s1) / PI + 0.5

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1
```

`PEXExtTMProjectionCylinderWC`

The infinite cylinder has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given coordinate sources which are directional (normal or - reflection), a direction vector (s0,s1,s2) is computed to determine a point on the interior of the cylinder. If the coordinate source is vertex coord, a ray perpendicular to the +Y axis through the vertex is computed to determine a point and its height on the interior of the cylinder. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
     (-arctan(s2/s0) / (2 * PI) +
        ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
```

)

if (coord_source == normal)
    t1 = arcsin(s1) / PI + 0.5
else
    t1 = y_coord

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1

`PEXExtTMProjectionLinearWC`

Parameters for two linear equations are defined a p0 and p1. Given the specified coordinate source, the values will be used directly as (s0,s1,s2) to linearly project the values by the equations listed below. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list. (It should be noted that "s3" is assumed to be one as shown in the equations below.)

t0 = s0 * p0[0] + s1 * p0[1] + s2 * p0[2] + 1 * p0[3]
t1 = s0 * p1[0] + s1 * p1[1] + s2 * p1[2] + 1 * p1[3]

It should be noted that the projection of anything other than vertices for this option will lead to unpredictable visual

It is assumed the vertex list of the specified primitive has sufficient space to insert the needed texture coordinates. If space (indicated by `PEXExtGAData`) or valid indices for texture coordinates do not exist, an error will be returned and the data will remain unchanged.

ERRORS:

None

SEE ALSO:

`PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordTriangleStrip`, `PEXExtTMCoordQuadrilateralMesh`

▼ Extended Compute Triangle Strip Texture Coordinates

NAME:

PEXExtTMCoordTriangleStrip - computes a projection of texture coordinates for a triangle strip

SYNTAX:

int PEXExtTMCoordTriangleStrip(PEXExtTMCoordData **tm_coord_data*,
unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*,
unsigned int *vertex_attributes*, int *color_type*,
PEXArrayOfFacetData *facet_data*, unsigned int *count*,
PEXExtArrayOfVertex *vertices*)

PARAMETERS:

tm_coord_data	A pointer to parameterization data for this primitive (See *Description* in `PEXExtTMCoordFillAreaSetWithData`).
vertex_fp_data_size	The number of floating point data values defined per vertex.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXExtGAData`).
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
facet_data	An array of facet data.
count	The number of vertices.
vertices	An array of vertices defining the triangle strip.

RETURNS:

Zero if successful; otherwise, one of the following:

`PEXExtBadPrimitive`	Normals are not valid.
`PEXExtBadValue`	Projection type is not valid.

`PEXExtBadValue`	Texture coordinate location not valid.
`PEXExtBadAlloc`	Internal allocation error has occurred.

DESCRIPTION:

This function computes a projection of texture coordinates onto a triangle strip and stores them in the specified vertex data fields.

To compute texture coordinates, the following fields in the *tm_coord_data* structure are initialized:

projection — Texture coordinate projection to perform (`PEXExtTMProjectionSphereWC`, `PEXExtTMProjectionCylinderWC`, `PEXExtTMProjectionLinearWC`)

matrix — If projection is either `PEXExtTMProjectionSphereWC` or `PEXExtTMProjectionCylinderWC`, this transform is applied to vertices and vertex normals after the *mc_transform* has been applied. It is used to orient the data relative to the projection object (sphere, or cylinder) as described.

p0, p1 — If projection is `PEXExtTMProjectionLinearWC`, then these elements contain linear projection equations. (See description below.)

coord_source — Source coordinate to compute projections specified with projection type above (`PEXExtTMCoordSourceVertexCoord` or `PEXExtTMCoordSourceVertexNormal`).

fp_data_index — Index within the vertex floating point data list in which to store the calculated texture coordinates. Space must already exist in the vertex floating point data list and the index must point to a valid location.

mc_transform — This transform is applied to vertices and vertex normals in model coordinates prior to computing the specified projection.

Additionally, for the given projections, the following information must be provided. For projections listed below, those which require a unit direction vector (s0,s1,s2) shall have one determined by the coordinate source selection previously mentioned. If `PEXExtTMCoordSourceVertexCoord` is selected, a ray is computed from the WC origin through the vertex to the interior of the projection object. If `PEXExtTMCoordSourceVertexNormal` is selected, the primitive's vertex normals (given or implicit) are used to determine the direction. The resulting values (t0,t1) are the texture coordinates stored at the requested index.

`PEXExtTMProjectionSphereWC`

The infinite sphere has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given the specified coordinate source, a direction vector (s0,s1,s2) is computed to determine a point on the interior of the sphere. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
    (-arctan(s2/s0) / (2 * PI) +
      ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
    )

t1 = arcsin(s1) / PI + 0.5

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1
```

`PEXExtTMProjectionCylinderWC`

The infinite cylinder has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given coordinate sources which are directional (normal or - reflection), a direction vector (s0,s1,s2) is computed to determine a point on the interior of the cylinder. If the coordinate source is vertex coord, a ray perpendicular to the +Y axis through the vertex is computed to determine a point and its height on the interior of the cylinder. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
    (-arctan(s2/s0) / (2 * PI) +
      ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
    )

if (coord_source == normal)
    t1 = arcsin(s1) / PI + 0.5
else
    t1 = y_coord

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2
```

Where (s0*s0 + s1*s1 + s2*s2) = 1

`PEXExtTMProjectionLinearWC`

Parameters for two linear equations are defined a p0 and p1. Given the specified coordinate source, the values will be used directly as (s0,s1,s2) to linearly project the values by the equations listed below. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list. (It should be noted that "s3" is assumed to be one as shown in the equations below.)

t0 = s0 * p0[0] + s1 * p0[1] + s2 * p0[2] + 1 * p0[3]
t1 = s0 * p1[0] + s1 * p1[1] + s2 * p1[2] + 1 * p1[3]

It should be noted that the projection of anything other than vertices for this option will lead to unpredictable visual

It is assumed the vertex list of the specified primitive has sufficient space to insert the needed texture coordinates. If space (indicated by `PEXExtGAData`) or valid indices for texture coordinates do not exist, an error will be returned and the data will remain unchanged.

ERRORS:

None

SEE ALSO:

`PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordQuadrilateralMesh`

▼ Extended Compute Quadrilateral Mesh Texture Coordinates

NAME:

PEXExtTMCoordQuadrilateralMesh - computes a projection of texture coordinates for a quadrilateral mesh

SYNTAX:

int PEXExtTMCoordQuadrilateralMesh(PEXExtTMCoordData **tm_coord_data*,
unsigned int *vertex_fp_data_size*,
unsigned int *facet_attributes*,
unsigned int *vertex_attributes*,
int *color_type*, PEXArrayOfFacetData *facet_data*,
unsigned int *col_count*, unsigned int *row_count*,
PEXExtArrayOfVertices *vertices*)

PARAMETERS:

tm_coord_data	A pointer to parameterization data for this primitive (See *Description* in `PEXExtTMCoordFillAreaSetWithData`).
vertex_fp_data_size	The number of floating point data values defined per vertex.
facet_attributes	A mask indicating the facet attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`).
vertex_attributes	A mask indicating the vertex attributes provided (`PEXGANone`, `PEXGAColor`, `PEXGANormal`, `PEXExtGAData`).
color_type	The type of color data provided (`PEXColorTypeIndexed`, `PEXColorTypeRGB`, `PEXColorTypeCIE`, `PEXColorTypeHSV`, `PEXColorTypeHLS`, `PEXColorTypeRGB8`, `PEXColorTypeRGB16`).
facet_data	An array of facet data.
col_count	The number of columns in the vertex array.
row_count	The number of rows in the vertex array.
vertices	A two-dimensional (row-major) array of vertices defining the quadrilateral mesh.

RETURNS:

Zero if successful; otherwise, one of the following:

`PEXExtBadPrimitive`	Normals are not valid.
`PEXExtBadValue`	Projection type is not valid.
`PEXExtBadValue`	Texture coordinate location not valid.
`PEXExtBadAlloc`	Internal allocation error has occurred.

DESCRIPTION:

This function computes a projection of texture coordinates onto a quadrilateral mesh and stores them in the specified vertex data fields.

To compute texture coordinates, the following fields in the *tm_coord_data* structure are initialized:

projection	Texture coordinate projection to perform (`PEXExtTMProjectionSphereWC`, `PEXExtTMProjectionCylinderWC`, `PEXExtTMProjectionLinearWC`)
matrix	If projection is either `PEXExtTMProjectionSphereWC` or `PEXExtTMProjectionCylinderWC`, this transform is applied to vertices and vertex normals after the *mc_transform* has been applied. It is used to orient the data relative to the projection object (sphere, or cylinder) as described.
p0, p1	If projection is `PEXExtTMProjectionLinearWC`, then these elements contain linear projection equations. (See description below.)
coord_source	Source coordinate to compute projections specified with projection type above (`PEXExtTMCoordSourceVertexCoord` or `PEXExtTMCoordSourceVertexNormal`).
fp_data_index	Index within the vertex floating point data list in which to store the calculated texture coordinates. Space must already exist in the vertex floating point data list and the index must point to a valid location.
mc_transform	This transform is applied to vertices and vertex normals in model coordinates prior to computing the specified projection.

Additionally, for the given projections, the following information must be provided. For projections listed below, those which require a unit direction vector (s0,s1,s2) shall have one determined by the coordinate source selection previously mentioned. If `PEXExtTMCoordSourceVertexCoord` is selected, a ray is computed from the WC origin through the vertex to the interior of the projection object. If `PEXExtTMCoordSourceVertexNormal` is selected, the primitive's vertex normals (given or implicit) are used to determine the direction. The resulting values (t0,t1) are the texture coordinates stored at the requested index.

`PEXExtTMProjectionSphereWC`

The infinite sphere has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given the specified coordinate source, a direction vector (s0,s1,s2) is computed to determine a point on the interior of the sphere. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
      (-arctan(s2/s0) / (2 * PI) +
         ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
      )

t1 = arcsin(s1) / PI + 0.5

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1
```

`PEXExtTMProjectionCylinderWC`

The infinite cylinder has (0,0,0) as its origin and the +Y axis as its axis of revolution. The texture seam sweeps from the +X axis in a counterclockwise direction [0..2PI]. Given coordinate sources which are directional (normal or - reflection), a direction vector (s0,s1,s2) is computed to determine a point on the interior of the cylinder. If the coordinate source is vertex coord, a ray perpendicular to the +Y axis through the vertex is computed to determine a point and its height on the interior of the cylinder. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list.

```
t0 = (s0 == 0.0) ? ((s2 > 0) ? 0.75 : 0.25) :
      (-arctan(s2/s0) / (2 * PI) +
         ((s0 < 0) ? 0.5 : (s0 > 0 && s2 > 0) ? 1 : 0)
      )

if (coord_source == normal)
      t1 = arcsin(s1) / PI + 0.5
```

else
    t1 = y_coord

-PI/2 <= arctan(r) <= PI/2
-PI/2 <= arcsin(r) <= PI/2

Where (s0*s0 + s1*s1 + s2*s2) = 1

`PEXExtTMProjectionLinearWC`

Parameters for two linear equations are defined as p0 and p1. Given the specified coordinate source, the values will be used directly as (s0,s1,s2) to linearly project the values by the equations listed below. The resulting texture coordinates (t0,t1) are stored in the vertex's floating point data list. (It should be noted that "s3" is assumed to be one as shown in the equations below.)

t0 = s0 * p0[0] + s1 * p0[1] + s2 * p0[2] + 1 * p0[3]
t1 = s0 * p1[0] + s1 * p1[1] + s2 * p1[2] + 1 * p1[3]

It should be noted that the projection of anything other than vertices for this option will lead to unpredictable visual

It is assumed the vertex list of the specified primitive has sufficient space to insert the needed texture coordinates. If space (indicated by `PEXExtGAData`) or valid indices for texture coordinates do not exist, an error will be returned and the data will remain unchanged.

ERRORS:

None

SEE ALSO:

`PEXExtTMCoordFillAreaSetWithData`, `PEXExtTMCoordSetOfFillAreaSets`, `PEXExtTMCoordTriangleStrip`

▼ Extended Create Filtered Texture Map

NAME:

PEXExtCreateFilteredTM - creates a prefiltered texture map from base map

SYNTAX:

int PEXExtCreateFilteredTM(int *domain*, PEXExtTMDomainData **domain_data*,
unsigned int *power_of_two_tm_required*,
unsigned int *square_tm_required*.
PEXExtTexelArray **base_map*,
PEXExtTexelArray ***texel_array*)

PARAMETERS:

domain	The texture map domain type.
domain_data	A pointer to domain information specifying the type of texture map to be created and the type of texel data.
power_of_two_tm_required	Indicates whether maps with dimensions of power of two is required (True, False).
square_tm_required	Indicates whether square maps is required (True, False).
base_map	The texel data of the desired filtered texture map's base level.
texel_array	Texel data created by this utility.

RETURNS:

Zero if successful; otherwise, one of the following:

`PEXExtBadDomain`	Domain is not valid.
`PEXExtBadDimension`	Dimension is not valid.
`PEXExtBadTexelType`	Texel type is not valid.
`PEXExtBadParameterization`	Parameterization type is not valid.
`PEXExtBadTMAlloc`	Memory allocation error.
`PEXExtBadTexelDimension`	Texel dimensions are not valid.
`PEXExtBadTexelData`	Texel data is not valid.

DESCRIPTION:

This function creates a prefiltered texture map and returns a list of texel arrays suitable for `PEXExtCreateTM`. The information contained in *domain* and *domain_data* will determine if a MipMap or other type of texture map is created from the base level data provided.

A value of zero in the domain data field *num_levels* indicates to the routine that it should generate as many levels as necessary to create a full MipMap pyramid or other type of supported map. If the value is non-zero, then the utility will create either the number of levels requested or all the levels for a full map, which ever is less. Upon exit, the *num_levels* field will be updated to the number of levels actually created by the utility.

Implementation constraints will be observed in the creation of the final texel data. If an implementation requires that maps be created with square dimensions and/or dimensions which are a power of two, the utility will apply the appropriate image sizing to the nearest correct dimension. When shrinking the image, the texels from the source image is sampled down using a box filter to create the destination image. When magnifying the image, the texels from the source image will be linearly interpolated to create the destination image.

Texel arrays allocated by this function are disposed of using the `PEXExtFreeFilteredTM` utility after they have been used by the `PEXExtCreateTM` function.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtCreateTM`, `PEXExtFreeFilteredTM`

▼ Extended Create Filtered Texture Map from Window

NAME:

PEXExtCreateFilteredTMFromWindow - creates a prefiltered texture map from a designated window

SYNTAX:

int PEXExtCreateFilteredTMFromWindow(Display **display*,
    int *domain*, PEXExtTMDomainData **domain_data*,
    unsigned int *power_of_two_tm_required*,
    unsigned int *square_tm_required*,
    unsigned int *luminance_channel_selector*,
    XID *base_color_window*,
    unsigned int *alpha_channel_selector*,
    XID *base_alpha_window*,
    PEXExtTexelArray ***texel_array*)

PARAMETERS:

display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
domain	The texture map domain type.
domain_data	A pointer to domain information specifying the type of texture map to be created and the type of texel data.
power_of_two_tm_required	Indicates whether maps with dimensions of power of two is required (True, False).
square_tm_required	Indicates whether square maps is required (True, False).
luminance_channel_selector	Luminance channel source selector. (`PEXExtChannelNTSCLuminance`, `PEXExtChannelRed`, `PEXExtChannelGreen`, `PEXExtChannelBlue`)
base_color_window	The X window identifier of the desired filtered texture map's base level color and/or luminance data.

alpha_channel_selector

Alpha channel source selector from alpha window, if relevant to texel type to be created. (`PEXExtChannelNTSCLuminance`, `PEXExtChannelRed`, `PEXExtChannelGreen`, `PEXExtChannelBlue`)

base_alpha_window

The X window identifier of the desired filtered texture map's base level alpha data.

texel_array Texel data created by this utility.

RETURNS:

Zero if successful; otherwise, one of the following:

`PEXExtBadXResource`	X resource ID is not valid.
`PEXExtBadDomain`	Domain is not valid.
`PEXExtBadTexelType`	Texel type is not valid.
`PEXExtBadTMAlloc`	Memory allocation error.
`PEXExtBadTMType`	Texture map type is not valid.
`PEXExtBadValue`	Luminance channel selector is not valid.
`PEXExtBadValue`	Alpha channel selector is not valid.

DESCRIPTION:

This function creates a prefiltered texture map and returns texel data suitable for `PEXExtCreateTM`. The information contained in *domain* and *domain_data* will determine if a MipMap or other type of texture map is created from the base level data provided.

The *base_color_window* parameter identifies a fully exposed and unobscured window from which to read texel data. If the texel type to create indicates alpha, then *base_alpha_window* identifies a fully exposed and unobscured window from which to read alpha texel data. If either window is obscured, then the results of the output texel array may be undefined.

A value of zero in the domain data field *num_levels* indicates to the routine that it should generate as many levels as necessary to create a full MipMap pyramid or other type of supported map. If the value is non-zero, then the utility will create either the number of levels requested or all the levels for a full map, whichever is less. Upon exit, the *num_levels* field will be updated to the number of levels actually created by the utility.

If the texel type specified in the description indicates luminance is included (i.e. the texel type is `PEXExtTexelLuminanceFloat`, `PEXExtTexel-LuminanceInt8`, `PEXExtTexelLuminanceInt16`, `PEXExtTexelLuminanceAlphaFloat`, `PEXExtTexelLuminanceAlphaInt8`, or `PEXExtTexelLuminanceAlphaInt16`), the *luminance_channel_selector* indicates the source of the luminance data. The channels available as source of the luminance data are dependent upon the depth or visual class of *base_color_window*.

If the texel type specified in the description indicates alpha is included (i.e. the texel type is `PEXExtTexelLuminanceAlphaFloat`, `PEXExtTexelLuminanceAlphaInt8`, `PEXExtTexelLuminanceAlphaInt16`, `PEXExtTexelRGBAlphaFloat`, `PEXExtTexelRGBAlphaInt8`, or `PEXExtTexelRGBAlphaInt16`), the *alpha_channel_selector* indicates the source of the alpha data. The channels available as source of the alpha data are dependent upon the depth or visual class of *base_alpha_window*.

Selecting `PEXExtChannelNTSCLuminance` results in calculating the luminance intensities from the active color map of the window resource as:

luminance = (red * 0.299 + green * 0.587 + blue * 0.114)

Selecting any other channel type directly uses the data in that channel as a source for luminance or alpha data.

Implementation constraints will be observed in the creation of the final texel data as indicated by *power_of_two_tm_required* and *square_tm_required*. If an implementation requires that maps be created with square dimensions and/or dimensions which are a power of two, the utility will apply the appropriate image sizing to the nearest correct dimension. When shrinking the image, the texels from the source image are sampled down using a box filter to create the destination image. When magnifying the image, the texels from the source image will be linearly interpolated to create the destination image.

Texel arrays allocated by this function are disposed of using the `PEXExtFreeFilteredTM` utility after they have been used by the `PEXExtCreateTM` function.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtCreateTM`, `PEXExtFreeFilteredTM`

▼ Extended Free Filtered Texture Map

NAME:

PEXExtFreeFilteredTM - frees up texture resources

SYNTAX:

void PEXExtFreeFilteredTM(int *domain*, PEXExtTMDomainData **domain_data*,
PEXExtTexelArray **texel_array*)

PARAMETERS:

domain	The texture map domain type.
domain_data	A pointer to texture map domain information.
texel_array	Texel data to be freed.

RETURNS:

None

DESCRIPTION:

This function frees storage returned by `CreateFilteredTM` utilities. Texel data created by these functions can be immediately freed by `PEXExtFreeFilteredTM` after using them with the `PEXExtCreateTM` function.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtCreateTM, PEXExtCreateFilteredTM`
`PEXExtCreateFilteredTMFromWindow`

Extended Count Output Commands

NAME:

PEXExtCountOCs - Extended Count Output Commands

SYNTAX:

unsigned long PEXExtCountOCs(int *float_format*, unsigned long *length*,
char **encoded_ocs*)

PARAMETERS:

float_format	The floating point format of the encoded output commands (PEXIEEE_754_32, PEXDEC_F_Floating, PEXIEEE_754_64, PEXDEC_D_Floating)
length	The length, in bytes, of the encoded output commands.
encoded_ocs	A pointer to the encoded output commands.

RETURNS:

The number of output commands represented in the encoded output commands.

DESCRIPTION:

This function has no visible effect. This function returns the number of output commands in the encoded list of output commands. A count of zero will be returned if the specified floating point format is not supported.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtDecodeOCs, PEXExtEncodeOCs, PEXExtFreeOCData`

▼ Extended Decode Output Commands

NAME:

PEXExtDecodeOCs - Extended Decode Output Commands

SYNTAX:

PEXExtOCData *PEXExtDecodeOCs(int *float_format*, unsigned long *oc_count*, unsigned long *length*, char **encoded_ocs*)

PARAMETERS:

float_format	The floating point format of the encoded output commands (PEXIEEE_754_32, PEXDEC_F_Floating, PEXIEEE_754_64, PEXDEC_D_Floating)
oc_count	The number of output commands represented in the encoded output commands.
length	The length, in bytes, of the encoded output commands.
encoded_ocs	A pointer to the encoded output commands.

RETURNS:

A pointer to the extended decoded output commands; a null pointer if unsuccessful or if zero output commands specified.

DESCRIPTION:

This function has no visible effect. Encoded output commands are passed in and the data typically passed as parameters to output attribute or primitive functions is returned in memory allocated by PEXlib. PEXExtFreeOCData should be called to deallocate the memory.

A null pointer will be returned if the specified floating point format is not supported.

Any text or annotation text primitives are returned as encoded text or encoded annotation text.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtEncodeOCs,PEXExtFreeOCData`

Extended Encode Output Commands

NAME:

PEXExtEncodeOCs - Extended Encode Output Commands

SYNTAX:

char *PEXExtEncodeOCs(int *float_format*, unsigned long *oc_count*,
PEXExtOCData **oc_data*, unsigned long **length_return*)

PARAMETERS:

float_format	The floating point format of the encoded output commands (PEXIEEE_754_32, PEXDEC_F_Floating, PEXIEEE_754_64, PEXDEC_D_Floating)
oc_count	The number of output commands to be encoded.
oc_data	An array of the extended output command data.
length_return	Returns the length, in bytes, of the encoded output commands.

RETURNS:

A pointer to the encoded output commands; a null pointer if unsuccessful or if zero output commands specified.

DESCRIPTION:

This function has no visible effect. The data typically passed as parameters to output attribute or primitive functions is passed in and encoded into protocol formatted output commands. The encoded data is returned in memory allocated by PEXlib. XFree should be called to deallocated the memory.

A null pointer will be returned if the specified floating point format is not supported.

Any text or annotation text primitives must be specified as encoded text or encoded annotation text.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtDecodeOCs`

▼ Extended Free Output Commands Data

NAME:

PEXExtFreeOCData - Extended Deallocate OC Data

SYNTAX:

void PEXExtFreeOCData(unsigned long *count*, PEXExtOCData **oc_data*)

PARAMETERS:

count	The number of output commands.
oc_data	An array of the extended output command data.

RETURNS:

None

DESCRIPTION:

This function has no visible effect. This function deallocates memory allocated by `PEXExtDecodeOCs` to hold decoded output command data.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

SEE ALSO:

`PEXExtDecodeOCs`

Extended Get Output Commands Size

NAME:

PEXExtGetSizeOCs - Extended Get OC Size

SYNTAX:

int PEXExtGetSizeOCs (int *float_format*, int *oc_count*, PEXExtOCData **oc_data*)

PARAMETERS:

float_format	The floating point format of the encoded output commands (PEXIEEE_754_32, PEXDEC_F_Floating, PEXIEEE_754_64, PEXDEC_D_Floating)
oc_count	The number of output commands.
oc_data	An array of the extended output command data.

RETURNS:

The size, in bytes, of the formatted output commands; zero if unsuccessful.

DESCRIPTION:

This function returns information about the size of the protocol for the extended output commands. An successful return value is possible if the specified floating point format is invalid or unsupported by PEXlib.

DATA STRUCTURES:

See also `PEXlib.h` and `PEXExtlib.h`

ERRORS:

None

▼ Extended Fetch Elements and Send

NAME:

PEXExtFetchElementsAndSend - Extended Fetch Elements and Send to Display

SYNTAX:

Status PEXExtFetchElementsAndSend (Display **src_display*,
PEXStructure *structure*, int *whence1*, long *offset1*,
int *whence2*, long *offset2*, Display **dst_display*,
XID *resource_id*, PEXOCRequestType *req_type*)

PARAMETERS:

src_display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
structure	The resource identifier of the structure.
whence1	A value of specifying, with *offset1*, the first limit of the range of elements to be fetched (`PEXBeginning`, `PEXCurrent`, `PEXEnd`).
offset1	The offset from *whence1* denoting the first limit of the range of elements to be fetched.
whence2	A value specifying, with *offset2*, the second limit of the range of elements to be fetched (`PEXBeginning`, `PEXCurrent`, `PEXEnd`).
offset2	The offset from *whence2* denoting the second limit of the range of elements to be fetched.
dst_display	A pointer to a display structure returned by a successful `XOpenDisplay` call.
resource_id	The resource identifier of the renderer or structure.
req_type	The request type for the output commands (`PEXOCRender`, `PEXOCStore`, `PEXOCRenderSingle` or `PEXOCStoreSingle`).

RETURNS:

Zero if unsuccessful, non-zero otherwise.

DESCRIPTION:

This function is like PEXExtFetchElements except that the list of output commands are not returned to the application but are sent directly to the specified destination display.

Calling this function is similar to calling PEXExtFetchElements, and then sending the returned list of output commands by calling PEXStartOC, PEXCopyBytesToOC and PEXFinishOC.

If the destination display does not support the same floating point format as the format PEXlib is using with the source display, and if PEXlib can not convert to a format supported by the destination display, function will return unsuccessfully.

Sending output commands to a structure whose editing mode is PEXStructureReplace, is not really useful. The behavior will be unpredictable unless a request type of PEXOCStoreSingle is used. And, if the request is PEXOCStoreSingle, each output command will simply replace the previous one sent. Applications should ensure that the structure's editing mode is PEXStructureInsert, when sending multiple output commands. If it is intended to replace multiple elements, the application can delete those elements first, and then insert the new ones.

DATA STRUCTURES:

See also PEXlib.h and PEXExtlib.h

ERRORS:

BadPEXStructure
The specified structure resource identifier is invalid.

BadValue
A specified value for *whence* parameter is invalid.

SEE ALSO

PEXExtFetchElements

Programming and Examples 6

This chapter describes general programming issues relating to Solaris PEX PEXlib, and code example programs are provided at the end of the chapter.

Programming Notes

When you install Solaris PEX, the files you need for building and running PEXlib application are installed by default in subdirectories under `/usr/openwin`. Figure 6-1 shows the default locations of these files:

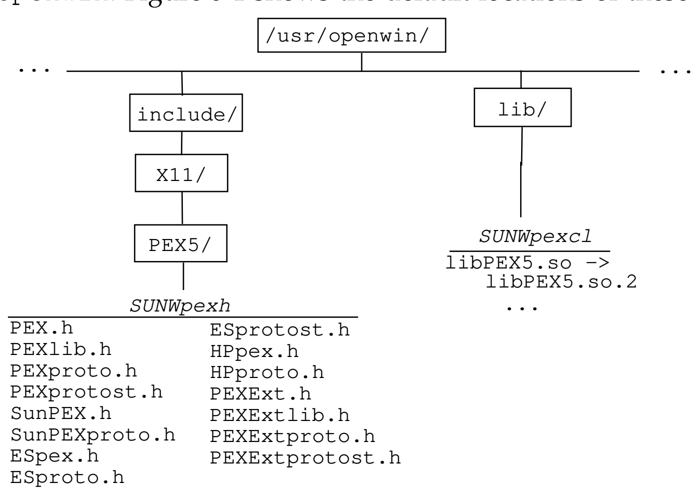

Figure 6-1 PEXlib Install Files Directory Tree

Library

You should link PEXlib programs you develop with the PEXlib shared-object library, `libPEX5.so` (as shown in Figure 6-1).

The `libPEX5.so` library is normally installed in `/usr/openwin/lib`, as shown. When you run PEXlib applications, make sure that this directory is included in your value of the environment variable `LD_LIBRARY_PATH`. For example:

```
% env | grep LD_LIBRARY_PATH
/usr/lib
% setenv LD_LIBRARY_PATH /usr/openwin/lib:$LD_LIBRARY_PATH
```

Header Files

When you write PEXlib applications, include the header files that define PEX data structures and constants. These files are installed by default in `/usr/openwin/include/X11/PEX5`:

- `ESpex.h`
- `ESproto.h`
- `ESprotost.h`
- `HPpex.h`
- `HPproto.h`
- `PEX.h`
- `PEXlib.h`
- `PEXproto.h`
- `PEXprotost.h`
- `PEXExt.h`
- `PEXExtlib.h`
- `PEXExtproto.h`
- `PEXExtprotost.h`
- `SunPEX.h`

- `SunPEXproto.h`

For simple PEXlib programs, you probably need to include only the header file, `PEXlib.h`. For programs using Escapes, GDPs, or GSEs, you might need to include one or more of the last eleven header files listed above. For more information about escapes, GDPs, and GSEs, please see Table 4-25 through Table 4-28 in Chapter 4, "Functional Description."

Online Reference Manual Pages

The Solaris PEX package includes the Solaris PEXlib man pages which describe the syntax for using the functions in the PEX Graphics Library. These man pages are located in `/opt/SUNWsdk/2.5/pex/man/man3`.

Example Programs

The Solaris PEX package includes several example programs for PEXlib, including example programs from the *PEXlib Programming Manual.* By default, these programs are installed in `/opt/SUNWsdk/2.5/pex/examples`:

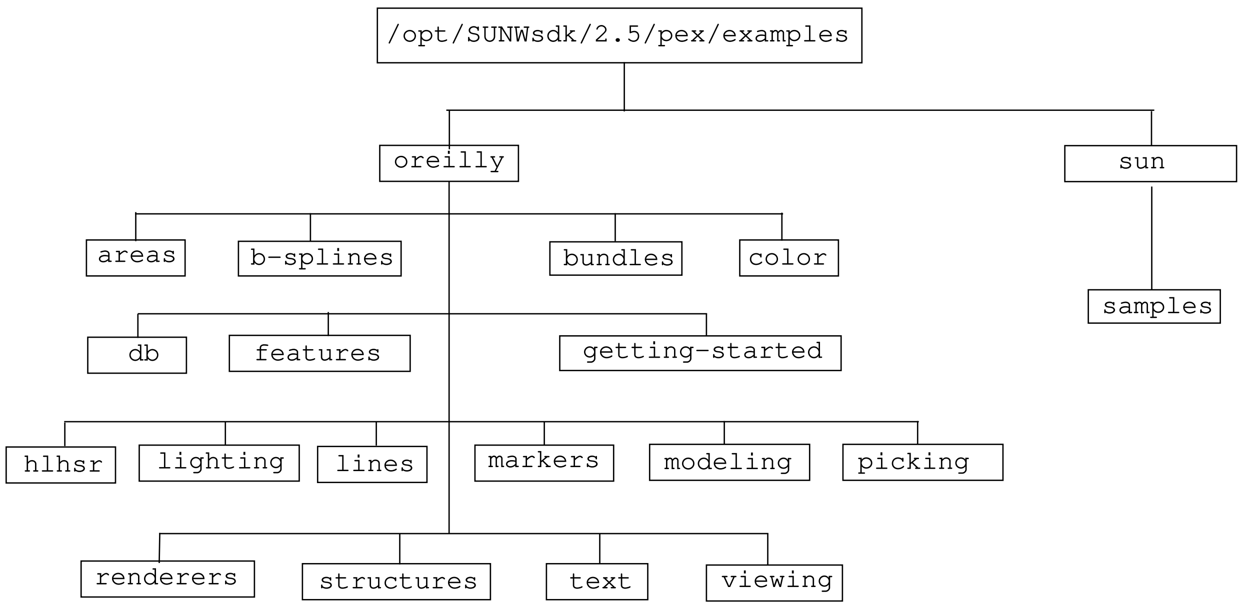

Figure 6-2 Example Programs Directory Tree

The example programs, as installed by default, are owned by the super user, root. To build these example programs, you must do either of the following:

- Make yourself "root" on your system
- Make modifications to makefiles and file protections and ownership in a copy of the directory tree or in place.

This chapter only shows how to build the example programs as root. Contact your system administrator for assistance, if necessary.

O'Reilly Examples

These are the examples from the *PEXlib Programming Manual*. To build them, `cd` to the `oreilly` directory, and issue the `make` command:

```
% cd /opt/SUNWsdk/2.5/pex/examples/oreilly
% make
```

This command makes a library of utilities, `libbook.a`, that is used by example programs under this subdirectory, and then makes the example programs.

Sun Examples

To build the Sun examples, `cd` to the `sun` directory, and issue the `make` command:

```
% cd /opt/SUNWsdk/2.5/pex/examples/sun
% make
```

Color Cube Performance Utility Example

For this example, you need only to `cd` to the `sun` subdirectory and issue the `make` command:

```
% cd /opt/SUNWsdk/2.5/pex/examples/sun
% make
```

This section discusses in detail the color cube performance utility example. This program is an example for how to work around a limitation of the Sun XGL library as used by Solaris PEX, and attain the best performance for 8-bit color windows on Sun platforms that support 8-bit visuals. To do this, the program ensures that there is a suitable colormap installed in this session on your server.

To ensure the best possible performance on 8-bit PseudoColor visuals on Sun frame buffers, you can run this example program before you run any other PEX applications. Or you can include similar code in the application you develop.

This example creates a colormap and allows this colormap to be shared with other applications running on the same server. The program is designed to accommodate two visual depths: 8-bit and 24-bit.

The call structure of the program is shown in Figure 6-3 as follows:

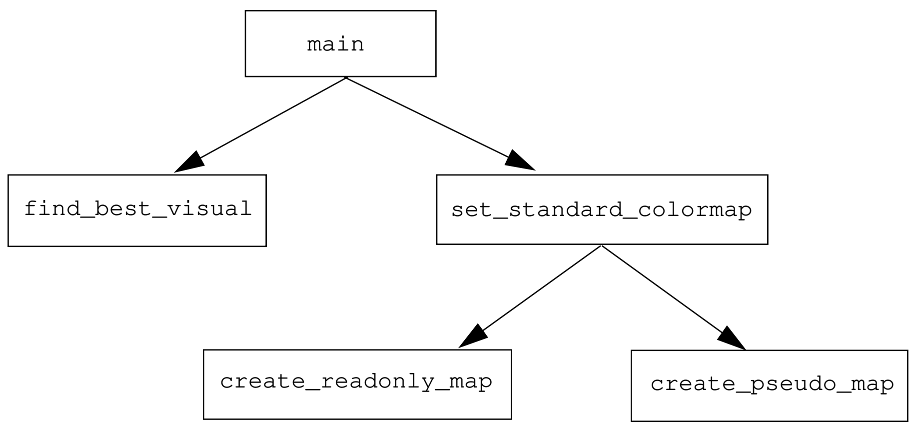

Figure 6-3 Call Structure

The command `main` determines the best visual to use by calling the subroutine `find_best_visual`. Then, it creates a color map based on that visual by calling the subroutine `set_standard_colormap`.

main

Code Example 6-1 Main Example

```
main(argc, argv)
int argc;
char *argv[];
{
    Display                     *dpy;
    XVisualInfo                 vis_info;

    dpy = XOpenDisplay();

    if ( !dpy ) exit(1);

    /* Determine the best visual to use.
     */
    if (!find_best_visual( dpy, &vis_info )) {
        fprintf(stderr, "Cannot find the best visual\n");
        exit(1);
    }

    /* create a standard colormap based on the chosen visual
     */
    if (!set_standard_colormap(dpy, &vis_info)){
        fprintf(stderr, "Cannot set standard colormap\n");
        exit(1);
    }

    /* set the close down mode to permanent so that the color
       resource will not be freed until the standard colormap
       property is deleted
     */
    XSetCloseDownMode(dpy, RetainPermanent);

    XSync(dpy);

}
```

find_best_visual

This routine is designed to handle only visuals of depth 8- or 24-bits. This routine assumes that a PseudoColor visual is best for 8-bit visuals and TrueColor for 24-bit visuals. It also assumes that `DefaultScreen` for your device has the same visual depth as you will be using; however, this may not be the case for devices with multiple visual depths. You might want to modify this subroutine to suit your own needs.

Code Example 6-2 Find the Best Visual Example

```
/* find the best visual
 */
static int
find_best_visual( dpy, chosen_vis )
    Display             *dpy;
    XVisualInfo         *chosen_vis;
{
    int                 err = 1;
    XVisualInfo         vinfo;

    /* Determine the best visual available. The best visual
       is the one with the most colors and highest capabilities
       and supported by PEX.
       This example assumes that PseudoColor visual is
       the best one on 8 bits device, and TrueColor visual is
       the best one on 24 bits device.
     */

    if (XMatchVisualInfo(dpy, DefaultScreen(dpy), 24, TrueColor,
                            &vinfo))
        *chosen_vis = vinfo;
    else if (XMatchVisualInfo(dpy, DefaultScreen(dpy), 8,
                            PseudoColor, &vinfo))
        *chosen_vis = vinfo;
    else
        err = 0;

    return (err);
}
```

set_standard_colormap

This routine ensures that there is a suitable colormap on your server during this session. For 8-bit PseudoColor visuals, this colormap should have the property `XA_RGB_DEFAULT_MAP` and contain a 6x9x4 color cube. If not, it creates a colormap containing this color cube with that property.

For 24-bit TrueColor visuals, this colormap should have the property `XA_RGB_BEST_MAP`. `set_standard_colormap` calls the subroutine `create_pseudo_map` to create the colormap for 8-bit PseudoColor visuals, and the subroutine `create_readonly_map` for 24-bit TrueColor visuals.

Code Example 6-3 Create a Standard RGB Map Example *(1 of 3)*

```
/* create a standard RGB map.
 */
int
set_standard_colormap( dpy, vis_info)
    Display             *dpy;
    XVisualInfo         *vis_info;
{
    XStandardColormap   *stdcmaps, cmap_info, *s;
    int                 i, count;
    int                 st = 1, done = 0;

    switch ( vis_info->class ) {
    case PseudoColor: {

        /* first see if the property exists
         */
        if (XGetRGBColormaps(dpy, DefaultRootWindow(dpy),
                &stdcmaps, &count, XA_RGB_DEFAULT_MAP)) {

            /* find the property that matches the given visualid
             */
            for (i = 0, s = stdcmaps;
              (i < count) && (s->visualid != vis_info->visualid);
                 i++, s++)
                ;

            /* verify that the color cube is a 6x9x4 color cube.
               Sun PEX performs best with a 6x9x4 color cube on
               a PseudoColor visual
             */
            if (i != count) {
```

Code Example 6-3 Create a Standard RGB Map Example *(2 of 3)*

```
/* create a standard RGB map.
                if ((s->red_max == 5) && (s->green_max == 8) &&
                    (s->blue_max == 3) &&
                    ((s->red_mult < s->green_mult) &&
                      (s->green_mult < s->blue_mult)))
                    done = 1;
            }
            XFree((char *)stdcmaps);
        }

        /* if property doesn't exist for the given visualid
         */
        if (!done) {

            /* create a 6x9x4 color cube
             */
            st = create_pseudo_map(dpy, vis_info, 6, 9, 4,
&cmap_info);
            XChangeProperty(dpy, DefaultRootWindow(dpy),
                XA_RGB_DEFAULT_MAP, XA_RGB_COLOR_MAP, 32,
PropModePrepend,
                (unsigned char *)&cmap_info, 10);
        }
        break;
    }
    case TrueColor: {

        /* make a best_map if it does not already exist
         */
        XGetRGBColormaps(dpy, DefaultRootWindow(dpy),
                &stdcmaps, &count, XA_RGB_BEST_MAP);

        if (count == 0) {
            /* create a readonly map
             */
            st = create_readonly_map(dpy, vis_info, &cmap_info);
            XChangeProperty(dpy, DefaultRootWindow(dpy),
                XA_RGB_BEST_MAP, XA_RGB_COLOR_MAP, 32,
PropModePrepend,
                (unsigned char *)&cmap_info, 10);
        } else
            XFree((char *)stdcmaps);

        break;
```

Code Example 6-3 Create a Standard RGB Map Example *(3 of 3)*

```
/* create a standard RGB map.
    }
    default:
        break;
    }
    return st;
}
```

create_pseudo_colormap

This routine creates a colormap containing a 6x9x4 color cube, enabling the best performance for 8-bit PseudoColor visuals on Sun graphics devices.

Code Example 6-4 Create a Pseudo Colormap Example *(1 of 3)*

```
/* create a pseudo colormap
 */
int
create_pseudo_map(dpy, vis_info, nr, ng, nb, cmap_info)
    Display             *dpy;
    XVisualInfo         *vis_info;
    int                 nr, ng, nb;
    XStandardColormap   *cmap_info;
{
    int                 num_colors, st, i, j, k, p;
    unsigned long       *pixels;
    XColor              *colors, *color;
    XWindowAttributes   root_window_attrs;

    XGetWindowAttributes(dpy, DefaultRootWindow(dpy),
&root_window_attrs);

    cmap_info->colormap = root_window_attrs.colormap;
    cmap_info->visualid = vis_info->visualid;
    cmap_info->red_max = nr - 1;
    cmap_info->red_mult = 1;
    cmap_info->green_max = ng - 1;
    cmap_info->green_mult = nr;
    cmap_info->blue_max = nb - 1;
    cmap_info->blue_mult = nr * ng;

    num_colors = nr * ng * nb;
    pixels = (unsigned long *) malloc(num_colors
*sizeof(*pixels));
```

Code Example 6-4 Create a Pseudo Colormap Example *(2 of 3)*

```
/* create a pseudo colormap

    /* Try to allocate the color cells from the default
       colormap first. It it doesn't fit, then create a private
       colormap
     */
    st = XAllocColorCells(dpy, cmap_info->colormap, True,
                (unsigned long *)NULL, 0, pixels, num_colors);
    if (st == 0) {
        cmap_info->colormap = XCreateColormap(dpy,
            RootWindow(dpy, DefaultScreen(dpy)),
            vis_info->visual, AllocNone);
        st = XAllocColorCells(dpy, cmap_info->colormap, True,
                (unsigned long *)NULL, 0, pixels, num_colors);
        if (st == 0) {
            free(pixels) ;
            return 0;
        }
        cmap_info->killid = ReleaseByFreeingColormap;
    } else {

        /* ICCCM requires an arbitrary resource id for the
           the killid if the colormap resource is the default
colormap
         */
        cmap_info->killid = XCreateWindow(dpy,
                RootWindow(dpy, DefaultScreen(dpy)),
             1,1,1,1,0,0, InputOnly, vis_info->visual, 0, NULL);
        XDestroyWindow(dpy, cmap_info->killid);
    }

    cmap_info->base_pixel = pixels[0];
    free(pixels);

    colors = (XColor *) malloc(num_colors * sizeof(*colors));
    p = cmap_info->base_pixel;
    for (i = 0, color = colors; i < nr; i++) {
        for (j = 0; j < ng; j++) {
            for (k = 0; k < nb; k++) {
                color->flags = DoRed | DoGreen | DoBlue;
                color->pixel = cmap_info->base_pixel +
                                i * cmap_info->red_mult +
                                j * cmap_info->green_mult +
                                k * cmap_info->blue_mult;
```

Code Example 6-4 Create a Pseudo Colormap Example *(3 of 3)*

```
/* create a pseudo colormap

                color->red = i * 65535 / cmap_info->red_max;
                color->green = j * 65535 / cmap_info->green_max;
                color->blue = k * 65535 / cmap_info->blue_max;
                ++color;
            }
        }
    }

    XStoreColors(dpy, cmap_info->colormap, colors, num_colors);
    free(colors);

    return 1;
}
```

create_readonly_map

This routine creates a colormap for 24-bit TrueColor visuals.

Code Example 6-5 Create a Readonly Map Example *(1 of 2)*

```
/* create a readonly map
 */
int
create_readonly_map(dpy, vis_info, cmap_info)
    Display             *dpy;
    XVisualInfo         *vis_info;
    XStandardColormap   *cmap_info;
{
    cmap_info->colormap = XCreateColormap(dpy,
            RootWindow(dpy, DefaultScreen(dpy)),
            vis_info->visual, AllocNone);

    cmap_info->base_pixel = 0;
    cmap_info->visualid = vis_info->visualid;

    cmap_info->red_max = vis_info->red_mask;
    cmap_info->red_mult = 1;
    while (!(cmap_info->red_max & 0x01)) {
        cmap_info->red_max >>= 1;
        cmap_info->red_mult <<= 1;
    }
    cmap_info->green_max = vis_info->green_mask;
```

Code Example 6-5 Create a Readonly Map Example *(2 of 2)*

```
/* create a readonly map
    cmap_info->green_mult = 1;
    while (!(cmap_info->green_max & 0x01)) {
        cmap_info->green_max >>= 1;
        cmap_info->green_mult <<= 1;
    }
    cmap_info->blue_max = vis_info->blue_mask;
    cmap_info->blue_mult = 1;
    while (!(cmap_info->blue_max & 0x01)) {
        cmap_info->blue_max >>= 1;
        cmap_info->blue_mult <<= 1;
    }

    cmap_info->killid = ReleaseByFreeingColormap;
    return 1;
}
```