PEXlib Specification and C Language Binding

Jeff Stevenson
Chief Architect and Document Editor
Hewlett-Packard Company

Version 5.1

November 5, 1992

Acknowledgements

On behalf of the MIT X Consortium, we gratefully acknowledge and give thanks to the many who have participated in the design, definition and implementation of PEXlib.

To Digital Equipment Corporation and Sony Microsystems, Inc., we give thanks for their generous donation of prototypical implementations to the MIT X Consortium for use in this effort. And to SHOGraphics and International Business Machines, thanks for donation of additional implementation effort. And last, but certainly not least, to Ralph Mor (X Consortium), thanks for completing the PEXlib sample implementation.

Also to Digital Equipment Corporation, thanks for their contribution of initial documentation which was leveraged in producing this document.

Thanks to Tom Gaskins (Sound Software Design) and Jan Hardenbergh (Oki Advanced Products Division) for their work in defining the utilities in *Chapter 16*.

Thanks to the many who provided timely review of document drafts, including Tom Gaskins (Sound Software Design), Ralph Mor (X Consortium), Bob Scheifler (Director of the X Consortium), John Waitz (Hewlett-Packard Co.) and many others.

Finally, we acknowledge the contribution of those involved in the design and definition of the PEX protocol, without which, PEXlib would not exist.

Chapter 1

Introduction to PEXlib

PEXlib is a programmer's interface to the PEX protocol. PEX is an extension to the X window system for supporting three-dimensional graphics.

The application can query the existence of the PEX server extension by calling **XQueryExtension** using the name, **PEX_NAME_STRING**, defined in PEXlib include files.

To begin using PEXlib, the application must initialize PEXlib by calling **PEXInitialize** before calling any PEXlib function which takes a display argument. Functions which do not take a display argument may be called prior to calling **PEXInitialize. PEXInitialize** must be called for each display connection the application will use. Failure to call **PEXInitialize** will result in undefined behavior for functions which take a display argument. Termination of PEXlib occurs implicitly at either program end or **XCloseDisplay**.

There are four other functions that should be called during initialization to determine what features and parameters are supported by each particular PEX server implementation. These functions are **PEXGetExtensionInfo**, **PEXGetImpDepConstants**, **PEXGetEnumTypeInfo**, and **PEXMatchRenderingTargets**. **PEXGetExtensionInfo** returns information about the PEX server extension such as protocol version, vendor string, release number and subset support. **PEXGetImpDepConstants** returns information about supported implementation-dependent constants. **PEXGetEnumTypeInfo** returns information about supported enumerated types. **PEXMatchRenderingTargets** returns information about supported drawable types.

Some PEXlib functions allocate memory on behalf of the application. Memory allocation often occurs in the case of inquiry functions where the size of data to be returned is unknown. Applications must use **XFree** to deallocate memory that has been allocated for them by PEXlib, unless a specific function is provided to free memory allocated for particular PEXlib objects. Where necessary, the specific function for deallocation is noted in the description of the function which allocates the memory. Memory allocated by PEXlib is considered to be 'owned' by PEXlib; the application must not modify pointers to memory returned by the library nor attempt to free such memory except by the specified interfaces. Data structure fields other than pointers to memory can be changed without consequence.

PEXlib is designed as an extension to Xlib using the same transport and error mechanisms as Xlib. This includes all requests, replies, events and errors. This means synchronization (e.g. **XSync** and **XSynchronize**) will have the same effect on PEXlib requests as they do on Xlib requests. PEX events also are treated in the same way as X events (e.g. use **XNextEvent**).

PEXlib functions usually do not check for invalid parameters. Applications should be careful to pass correct data to PEXlib. Incorrect values sent to the PEX server extension will cause an error event to be returned to the client. These errors are asynchronous, and so are difficult to trace to a particular Xlib or PEXlib call unless using **XSync** or **XSynchronize**. Note these functions should be used with prudence, since they slow client programs significantly.

By default, when an error event is returned by the server, Xlib prints a message and the application does not continue. The application can register its own error handler by calling **XSetErrorHandler**.

The floating point format used by PEXlib is the format native to the architecture of the client machine (i.e. the machine on which the application executes). All floating point values are expected to be in that native format, with a few exceptions for functions which allow the application to specify the floating point format. Those functions which allow the application to specify a floating point format deal primarily with application data that is already in protocol format, or data that the application wants to be encoded into protocol format for its own use.

When the server to which PEXlib is sending data does not support the floating point format native to PEXlib, PEXlib will choose an appropriate format which is supported by the server and convert into that format when formatting protocol to send to that server. In cases where PEXlib must choose a non-native floating point format, the application can determine the format chosen by calling **PEXGetProtocolFloatFormat**. This is useful with those functions dealing with protocol formatted data to help avoid unnecessary format conversions when sending the data to the PEX server.

One other function is defined in this opening chapter. **PEXGetSizeOCs** is a utility function for determining the protocol formatted size of output commands. If the output command(s) plus the request header is larger than the maximum request size supported by a particular server (see **XMaxRequestSize**), a **BadLength** error will be generated.

The PEXlib specification currently defines function bindings and include files for the C language only. The specification is intended to work for both ANSI C and Kernighan and Ritchie (K&R) C compilers.

The <**X11/PEX5/PEX.h**> and <**X11/PEX5/PEXlib.h**> include files contain constant definitions, macros, and data structures used by PEXlib. <**X11/PEX5/PEXlib.h**> automatically includes <**X11/PEX5/PEX.h**>. Some data structures have fields named "reserved". These fields are necessary to align the data correctly for efficient transfer to the protocol transport buffer. Any values assigned to these fields will be ignored.

The library is specified in the compilation command as "-lPEX5".

The utility functions in *Chapter 16* use math functions which create a dependency on the math library. This means that all PEXlib applications need to be linked with the math library, typically specified in the compilation command as "-lm".

Sincere effort has been made to adhere to the following conventions:

- PEXlib uses mixed case for external symbols. PEXlib leaves all lower-case for user variables and all upper-case for user macros, per existing convention. Where acronyms are used in a symbol, all characters of the acronym are upper-case, e.g. BF, HLHSR, LUT, OC, PC.
- All PEXlib functions, symbols, macros, user-visible data structures or anything a user might dereference begin with "PEX". A single exception to this is the error constants which are prefixed with "BadPEX" in order to be consistent with Xlib.
- The beginnings (i.e. immediately following the "PEX" prefix) of all function names and symbols are capitalized.
- All fields in a data structure are in lower-case. Compound words are constructed with underscores. This convention overrides the capitalization of acronyms, for data structure field names only.
- The display argument, where needed, is always first in the argument list.
- All resource arguments, where needed, occur at the beginning of the argument list but after the display argument. Where a drawable resource is needed, it will be listed before other resource arguments.
- Source arguments always precede destination arguments.
- Where a mask is specified, the mask always precedes the pointer to the structure.
- Where a count is specified, the count always precedes the pointer to the list.
- Min always precedes max.
- Coordinates are specified in x, y or x, y, z order.
- Color triplets are listed in r,g,b; h,s,v; h,l,s; u,v,l; order as appropriate for the color type.

- PEXlib defines constants for the user consistent with the PEX protocol. True and False are already defined by Xlib, and are consistent with PEX protocol.

PEXFreeEnumInfo - Free Memory Allocated for Enumerated Type Information

Synopsis

void PEXFreeEnumInfo(unsigned long *count*, unsigned long **info_count*, PEXEnumTypeDesc **enum_info*)

Arguments

count	The number of enumerated types.
info_count	An array of counts. This corresponds to the info_count_return parameter in **PEXGetEnumTypeInfo**.
enum_info	An array of enumerated type descriptors. This corresponds to the enum_info_return parameter of **PEXGetEnumTypeInfo**.

Returns

None

Description

PEXFreeEnumInfo frees memory allocated by PEXlib for the return value and return parameter in **PEXGetEnumTypeInfo**.

Errors

None

See Also

PEXGetEnumTypeInfo

PEXGetEnumTypeInfo - Get Enumerated Type Information

Synopsis

Status PEXGetEnumTypeInfo(Display **display*, Drawable *drawable*, unsigned long *count*, int **enum_types*, unsigned long *item_mask*, unsigned long ***info_count_return*, PEXEnumTypeDesc ***enum_info_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
count	The number of enumerated types.
enum_types	A list of enumerated types for which information is to be returned. (See the Description for valid values.)
item_mask	A mask indicating the data to be returned for each enumerated type. (See the Description for valid values.)
info_count_return	Returns an array of counts. For each enumerated type, there is an entry specifying the number of descriptors in the return value array.
enum_info_return	Returns an array of enumerated type descriptors containing the enumerated type information.

Returns

Zero if unsuccessful; non-zero otherwise.

Description

PEXGetEnumTypeInfo allows the application to inquire the supported values for each enumerated type. It returns a descriptor for each supported value of each specified enumerated type requested. These values will be valid for all drawables having the same root window and depth as the specified drawable.

The standard PEX enumerated types are:

PEXETATextStyle
PEXETColorApproxModel
PEXETColorApproxType
PEXETColorType
PEXETCurveApproxMethod
PEXETDisplayUpdateMode
PEXETEscape
PEXETFloatFormat
PEXETGDP2D
PEXETGDP
PEXETGSE
PEXETHatchStyle
PEXETHLHSRMode
PEXETInteriorStyle
PEXETLightType
PEXETLineType
PEXETMarkerType
PEXETModelClipOperator
PEXETParaSurfCharacteristics
PEXETPickAllMethod
PEXETPickDeviceType

PEXETPickOneMethod
PEXETPolylineInterpMethod
PEXETPromptEchoType
PEXETReflectionModel
PEXETRenderingColorModel
PEXETSurfaceApproxMethod
PEXETSurfaceEdgeType
PEXETSurfaceInterpMethod
PEXETTrimCurveApproxMethod

Enumerated type descriptors are made up of an index field and a mnemonic field. The index field contains the numeric value of the supported type, and the mnemonic field contains a string describing the type.

The item mask specifies which of the fields is returned in each enumerated type descriptor. Defined values for the item mask are **PEXETCounts**, **PEXETIndex**, **PEXETMnemonic** and **PEXETAll**. If the item mask is **PEXETCounts**, no descriptor values are returned; only the counts will be returned. If the item mask is **PEXETIndex**, the supported index values are returned in addition to the counts. If the item mask is **PEXETMnemonic**, the mnemonic strings describing the supported values are returned in addition to the counts. If the item mask is **PEXETAll**, both the supported index values and the mnemonic strings are returned in addition to the counts.

PEXlib allocates memory for the returned array of counts and for the return value array of enumerated type descriptors. **PEXFreeEnumInfo** should be called to deallocate this memory.

The following are the standard enumerated type index, mnemonic pairs which may be returned.

PEXETATextStyle

PEXATextNotConnected, **PEXETMATextNotConnected**
PEXATextConnected, **PEXETMATextConnected**

PEXETColorApproxModel

PEXColorApproxRGB, **PEXETMColorApproxRGB**
PEXColorApproxCIE, **PEXETMColorApproxCIE**
PEXColorApproxHSV, **PEXETMColorApproxHSV**
PEXColorApproxHLS, **PEXETMColorApproxHLS**
PEXColorApproxYIQ, **PEXETMColorApproxYIQ**

PEXETColorApproxType

PEXColorSpace, **PEXETMColorSpace**
PEXColorRange, **PEXETMColorRange**

PEXETColorType

PEXColorTypeIndexed, **PEXETMColorTypeIndexed**
PEXColorTypeRGB, **PEXETMColorTypeRGB**
PEXColorTypeCIE, **PEXETMColorTypeCIE**
PEXColorTypeHSV, **PEXETMColorTypeHSV**
PEXColorTypeHLS, **PEXETMColorTypeHLS**
PEXColorTypeRGB8, **PEXETMColorTypeRGB8**
PEXColorTypeRGB16, **PEXETMColorTypeRGB16**

PEXETCurveApproxMethod

PEXApproxImpDep, implementation-dependent string
PEXApproxConstantBetweenKnots, **PEXETMApproxConstantBetweenKnots**
PEXApproxWCChordalSize, **PEXETMApproxWCChordalSize**
PEXApproxNPCChordalSize, **PEXETMApproxNPCChordalSize**
PEXApproxDCChordalSize, **PEXETMApproxDCChordalSize**
PEXCurveApproxWCChordalDev, **PEXETMCurveApproxWCChordalDev**

PEXCurveApproxNPCChordalDev, **PEXETMCurveApproxNPCChordalDev**
PEXCurveApproxDCChordalDev, **PEXETMCurveApproxDCChordalDev**
PEXApproxWCRelative, **PEXETMApproxWCRelative**
PEXApproxNPCRelative, **PEXETMApproxNPCRelative**
PEXApproxDCRelative, **PEXETMApproxDCRelative**

PEXETDisplayUpdateMode

PEXVisualizeEach, **PEXETMVisualizeEach**
PEXVisualizeEasy, **PEXETMVisualizeEasy**
PEXVisualizeNone, **PEXETMVisualizeNone**
PEXSimulateSome, **PEXETMSimulateSome**
PEXVisualizeWhenever, **PEXETMVisualizeWhenever**

PEXETEscape

PEXEscapeSetEchoColor **PEXETMEscapeSetEchoColor**

PEXETFloatFormat

PEXIEEE_754_32, **PEXETMIEEE_754_32**
PEXDEC_F_Floating, **PEXETMDEC_F_Floating**
PEXIEEE_754_64, **PEXETMIEEE_754_64**
PEXDEC_D_Floating, **PEXETMDEC_D_Floating**

PEXETGDP2D

There are currently no standard 2D generalized drawing primitives.

PEXETGDP

There are currently no standard 3D generalized drawing primitives.

PEXETGSE

There are currently no standard generalized structure elements.

PEXETHatchStyle

There are currently no standard hatch styles.

PEXETHLHSRMode

PEXHLHSROff, **PEXETMHLHSROff**
PEXHLHSRZBuffer, **PEXETMHLHSRZBuffer**
PEXHLHSRPainters, **PEXETMHLHSRPainters**
PEXHLHSRScanline, **PEXETMHLHSRScanline**
PEXHLHSRHiddenLineOnly, **PEXETMHLHSRHiddenLineOnly**
PEXHLHSRZBufferID, **PEXETMHLHSRZBufferID**

PEXETInteriorStyle

PEXInteriorStyleHollow, **PEXETMInteriorStyleHollow**
PEXInteriorStyleSolid, **PEXETMInteriorStyleSolid**
PEXInteriorStylePattern, **PEXETMInteriorStylePattern**
PEXInteriorStyleHatch, **PEXETMInteriorStyleHatch**
PEXInteriorStyleEmpty, **PEXETMInteriorStyleEmpty**

PEXETLightType

PEXLightAmbient, **PEXETMLightAmbient**
PEXLightWCVector, **PEXETMLightWCVector**
PEXLightWCPoint, **PEXETMLightWCPoint**
PEXLightWCSpot, **PEXETMLightWCSpot**

PEXETLineType

PEXLineTypeSolid, **PEXETMLineTypeSolid**
PEXLineTypeDashed, **PEXETMLineTypeDashed**
PEXLineTypeDotted, **PEXETMLineTypeDotted**
PEXLineTypeDashDot, **PEXETMLineTypeDashDot**

PEXETMarkerType

PEXMarkerDot, **PEXETMMarkerDot**
PEXMarkerCross, **PEXETMMarkerCross**
PEXMarkerAsterisk, **PEXETMMarkerAsterisk**
PEXMarkerCircle, **PEXETMMarkerCircle**
PEXMarkerX, **PEXETMMarkerX**

PEXETModelClipOperator

PEXModelClipReplace, **PEXETMModelClipReplace**
PEXModelClipIntersection, **PEXETMModelClipIntersection**

PEXETParaSurfCharacteristics

PEXPSCNone, **PEXETMPSCNone**
PEXPSCImpDep, implementation-dependent string
PEXPSCIsoCurves, **PEXETMPSCIsoCurves**
PEXPSCMCLevelCurves, **PEXETMPSCMCLevelCurves**
PEXPSCWCLevelCurves, **PEXETMPSCWCLevelCurves**

PEXETPickAllMethod

PEXPickAllAll, **PEXETMPickAllAll**
PEXPickAllVisible, **PEXETMPickAllVisible**

PEXETPickDeviceType

PEXPickDeviceDCHitBox, **PEXETMPickDeviceDCHitBox**
PEXPickDeviceNPCHitVolume, **PEXETMPickDeviceNPCHitVolume**

PEXETPickOneMethod

PEXPickLast, **PEXETMPickLast**
PEXPickClosestZ, **PEXETMPickClosestZ**
PEXPickVisibleAny, **PEXETMPickVisibleAny**
PEXPickVisibleClosest, **PEXETMPickVisibleClosest**

PEXETPolylineInterpMethod

PEXPolylineInterpNone, **PEXETMPolylineInterpNone**
PEXPolylineInterpColor, **PEXETMPolylineInterpColor**

PEXETPromptEchoType

PEXEchoPrimitive, **PEXETMEchoPrimitive**
PEXEchoStructure, **PEXETMEchoStructure**
PEXEchoNetwork, **PEXETMEchoNetwork**

PEXETReflectionModel

PEXReflectionNone, **PEXETMReflectionNone**
PEXReflectionAmbient, **PEXETMReflectionAmbient**
PEXReflectionDiffuse, **PEXETMReflectionDiffuse**
PEXReflectionSpecular, **PEXETMReflectionSpecular**

PEXETRenderingColorModel

PEXRenderingColorModelImpDep, implementation-dependent string
PEXRenderingColorModelRGB, **PEXETMRenderingColorModelRGB**
PEXRenderingColorModelCIE, **PEXETMRenderingColorModelCIE**
PEXRenderingColorModelHSV, **PEXETMRenderingColorModelHSV**
PEXRenderingColorModelHLS, **PEXETMRenderingColorModelHLS**

PEXETSurfaceApproxMethod

PEXApproxImpDep, implementation-dependent string
PEXApproxConstantBetweenKnots, **PEXETMApproxConstantBetweenKnots**
PEXApproxWCChordalSize, **PEXETMApproxWCChordalSize**
PEXApproxNPCChordalSize, **PEXETMApproxNPCChordalSize**
PEXApproxDCChordalSize, **PEXETMApproxDCChordalSize**
PEXSurfaceApproxWCPlanarDev, **PEXETMSurfaceApproxWCPlanarDev**
PEXSurfaceApproxNPCPlanarDev, **PEXETMSurfaceApproxNPCPlanarDev**
PEXSurfaceApproxDCPlanarDev, **PEXETMSurfaceApproxDCPlanarDev**
PEXApproxWCRelative, **PEXETMApproxWCRelative**
PEXApproxNPCRelative, **PEXETMApproxNPCRelative**
PEXApproxDCRelative, **PEXETMApproxDCRelative**

PEXETSurfaceEdgeType

PEXSurfaceEdgeSolid, **PEXETMSurfaceEdgeSolid**
PEXSurfaceEdgeDashed, **PEXETMSurfaceEdgeDashed**
PEXSurfaceEdgeDotted, **PEXETMSurfaceEdgeDotted**
PEXSurfaceEdgeDashDot, **PEXETMSurfaceEdgeDashDot**

PEXETSurfaceInterpMethod

PEXSurfaceInterpNone, **PEXETMSurfaceInterpNone**
PEXSurfaceInterpColor, **PEXETMSurfaceInterpColor**
PEXSurfaceInterpDotProduct, **PEXETMSurfaceInterpDotProduct**
PEXSurfaceInterpNormal, **PEXETMSurfaceInterpNormal**

PEXETTrimCurveApproxMethod

PEXApproxImpDep, implementation-dependent string
PEXApproxConstantBetweenKnots, **PEXETMApproxConstantBetweenKnots**

Data Structures

```
typedef struct {
    PEXEnumTypeIndex    index;
    char                *descriptor;            /* null terminated string */
} PEXEnumTypeDesc;
```

Errors

BadDrawable The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported.

BadValue A specified enumerated type is invalid.

See Also

PEXFreeEnumInfo

PEXGetExtensionInfo - Get Extension Information

Synopsis

PEXExtensionInfo *PEXGetExtensionInfo(Display **display*)

Arguments

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

Returns

A pointer to the extension information; a null pointer if unsuccessful.

Description

PEXGetExtensionInfo allows an application program to inquire the extension information from a PEX server extension.

The major version number, minor version number, release number, vendor name, and subset information are returned. If the subset value is **PEXCompleteImplementation**, the extension is a full PEX implementation. If the subset value is **PEXImmediateMode**, the extension supports only the "immediate rendering" subset of PEX. If the subset value is **PEXStructureMode**, the extension supports only the "structure rendering" subset of PEX. If the subset value is (**PEXImmediateMode** & **PEXStructureMode**), the extension supports both the "immediate rendering" and the "structure rendering" subsets of PEX. If the subset value is **PEXWorkstationOnly**, the extension supports only the "PHIGS workstation" subset of PEX.

The memory returned by this function is private to PEXlib and must not be modified or freed by the application.

Data Structures

```
typedef struct {
    unsigned short      major_version;
    unsigned short      minor_version;
    unsigned long       release;
    unsigned long       subset_info;
    char                *vendor_name;
    int                 major_opcode;
    int                 first_event;
    int                 first_error;
} PEXExtensionInfo;
```

Errors

None

See Also

PEXGetImpDepConstants - Get Implementation Dependent Constants

Synopsis

Status PEXGetImpDepConstants(Display **display*, Drawable *drawable*, unsigned long *count*, unsigned short **names*, PEXImpDepConstant ***constants_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
count	The number of implementation-dependent constants.
names	An array of names of implementation-dependent constants to be returned. (See the Description for valid values.)
constants_return	Returns an array of implementation-dependent constants.

Returns

Zero if unsuccessful; non-zero otherwise.

Description

PEXGetImpDepConstants allows an application program to query one or more implementation-dependent constants.

A single integer or floating-point value is returned for each value requested. These values are returned in order, with one return value in constants for each item in names. The implementation-dependent constants returned are based on the values used for a drawable having the same root and depth as the drawable specified. PEXlib allocates memory for the returned constants. **XFree** should be called to deallocate the memory.

The following implementation-dependent constants are standard:

PEXIDBestColorApprox (integer) - Either **PEXColorApproxPowersOf2** or **PEXColorApproxAnyValues**, depending on whether there is a significant performance gain if the number of reds/greens/blues in the color approximation table entry is a power of two, so pixels can be composed using shifts and adds.

PEXIDDitheringSupported (integer) - Either **True** if the dithering hint in color approximation lookup tables is used to control dithering, or **False**, if the dithering hint in color approximation lookup tables is not used.

PEXIDDoubleBufferingSupported (integer) - Either **True** if the server supports double-buffering for workstation resources, or **False**, if the server does not supports double-buffering for workstation resources.

PEXIDMaxEdgeWidth (integer) - Width (in pixels) of widest edge that can be drawn.

PEXIDMaxHitsEventSupported (integer) - **True** if the server supports the **PEXMaxHitsReached** event, or **False**, if the server does not support the event.

PEXIDMaxLineWidth (integer) - Width (in pixels) of widest line or curve that can be drawn.

PEXIDMaxMarkerSize (integer) - Largest dimension (either height or width, in pixels) of largest marker that can be drawn. This maximum is exclusive of the marker type **PEXMarkerDot** which is always drawn as the smallest displayable point.

PEXIDMaxModelClipPlanes (integer) - Maximum number of modeling clipping planes may be defined.

PEXIDMaxNameSetNames (integer) - Maximum number of names allowed in a name set.

PEXIDMaxNonAmbientLights (integer) - Maximum number of non-ambient light sources that can be enabled at one time.

PEXIDMaxNURBOrder (integer) - Maximum non-uniform rational B-spline order supported.

PEXIDMaxTrimCurveOrder (integer) - Maximum order for trim curves.

PEXIDMinEdgeWidth (integer) - Width (in pixels) of thinnest edge that can be drawn.

PEXIDMinLineWidth (integer) - Width (in pixels) of thinnest line or curve that can be drawn.

PEXIDMinMarkerSize (integer) - Largest dimension (either height or width, in pixels) of smallest marker that can be drawn. This minimum is exclusive of the marker type **PEXMarkerDot** which is always drawn as the smallest displayable point.

PEXIDNominalEdgeWidth (integer) - Width (in pixels) of "standard" edge.

PEXIDNominalLineWidth (integer) - Width (in pixels) of "standard" line or curve.

PEXIDNominalMarkerSize (integer) - Largest dimension (either height or width, in pixels) of "standard" marker.

PEXIDNumSupportedEdgeWidths (integer) - Number of supported edge widths. A value of 0 indicates that all edge widths, including fractional widths, between the minimum and maximum edge width are supported.

PEXIDNumSupportedLineWidths (integer) - Number of supported line or curve widths. A value of 0 indicates that all line widths, including fractional widths, between the minimum and maximum line width are supported.

PEXIDNumSupportedMarkerSizes (integer) - Number of supported marker sizes. A value of 0 indicates that all marker sizes, including fractional values, between the minimum and maximum marker size are supported.

PEXIDTransparencySupported (integer) - Either **True** if the transmission coefficient is utilized in the reflectance calculations, or **False**, if the transmission coefficient is not utilized.

PEXIDChromaticityRedU (flt_point) - CIEYUV u chromaticity coefficient for the red channel of the (properly adjusted) display device.

PEXIDChromaticityRedV (flt_point) - CIEYUV v chromaticity coefficient for the red channel of the (properly adjusted) display device.

PEXIDLuminanceRed (flt_point) - CIEYUV luminance value for the red channel of the (properly adjusted) display device.

PEXIDChromaticityGreenU (flt_point) - CIEYUV u chromaticity coefficient for the green channel of the (properly adjusted) display device.

PEXIDChromaticityGreenV (flt_point) - CIEYUV v chromaticity coefficient for the green channel of the (properly adjusted) display device.

PEXIDLuminanceGreen (flt_point) - CIEYUV luminance value for the green channel of the (properly adjusted) display device.

PEXIDChromaticityBlueU (flt_point) - CIEYUV u chromaticity coefficient for the blue channel of the (properly adjusted) display device.

PEXIDChromaticityBlueV (flt_point) - CIEYUV v chromaticity coefficient for the blue channel of the (properly adjusted) display device.

PEXIDLuminanceBlue (flt_point) - CIEYUV luminance value for the blue channel of the (properly adjusted) display device.

PEXIDChromaticityWhiteU (flt_point) - CIEYUV u chromaticity coefficient for the reference white of the (properly adjusted) display device.

PEXIDChromaticityWhiteV (flt_point) - CIEYUV v chromaticity coefficient for the reference white of the (properly adjusted) display device.

PEXIDLuminanceWhite (flt_point) - CIEYUV luminance value for the reference white of the (properly adjusted) display device.

Data Structures

```
typedef union {
    unsigned long       integer;
    float               flt_point;
} PEXImpDepConstant;
```

Errors

BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported.
BadValue	A specified implementation-dependent constant name is invalid.

See Also

PEXInitialize - Initialize PEXlib display connection

Synopsis

int PEXInitialize(Display **display*, PEXExtensionInfo ***info_return*, int *length*, char **error_string*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
info_return	Returns a pointer to the extension information; if available (see the Description).
length	The length, in bytes, of the memory allocated for the error string.
error_string	A pointer to memory allocated for the error string.

Returns

Zero if successful; otherwise, one of the following return values:

PEXBadExtension
PEXBadProtocolVersion
PEXBadFloatConversion
PEXBadLocalAlloc

Description

PEXInitialize initializes PEXlib for the specified display.

PEXInitialize can be called multiple times; subsequent calls will result in the same return value as the first call.

Standard **PEXInitialize** failure return values are:

PEXBadExtension - the PEX server extension does not exist,
PEXBadProtocolVersion - the PEX server extension does not support a compatible protocol version,
PEXBadFloatConversion - the PEX server extension does not support a protocol floating point format compatible with PEXlib's native format or a format to which PEXlib supports conversion, or
PEXBadLocalAlloc - PEXlib client-side allocation failed.

If **PEXInitialize** is successful (return value is zero), or if the return value is **PEXBadProtocolVersion**, a pointer to the extension information is returned in info_return. Otherwise, a NULL pointer is returned in info_return. The extension information is private to PEXlib and must not be modified or freed by the application.

The error string parameter specifies an address to memory allocated by the application. The constant **PEXErrorStringLength** is defined as a guideline for the size to allocate for the error string. If no failure occurred, the memory addressed by the error string parameter will be unchanged. If a failure does occur, an error string giving further information about the failure will be copied into this memory (up to the maximum specified by the length parameter).

The actual string returned is implementation dependent, and is provided primarily for convenience in printing an error message for the application's end-user.

Data Structures

```
typedef struct {
    unsigned short      major_version;
    unsigned short      minor_version;
    unsigned long       release;
```

```
    unsigned long       subset_info;
    char                *vendor_name;
    int                 major_opcode;
    int                 first_event;
    int                 first_error;
} PEXExtensionInfo;
```

Errors

None

See Also

PEXGetExtensionInfo, **PEXGetEnumTypeInfo**, **PEXGetImpDepConstants**, **PEXMatchRenderingTargets**

PEXMatchRenderingTargets - Return Information about Supported Rendering Targets

Synopsis

Status PEXMatchRenderingTargets(Display **display*, Drawable *drawable*, int *depth*, int *type*, Visual **visual*, unsigned long *max_targets*, unsigned long **count_return*, PEXRenderingTarget ***targets_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The drawable indicates which screen the application is interested in.
depth	The depth of interest.
type	The drawable type of interest (**PEXAnyDrawable**, **PEXWindowDrawable**, **PEXPixmapDrawable**, **PEXBufferDrawable**).
visual	The visual of interest.
max_targets	The maximum number of targets to return.
count_return	Returns the actual number of targets in the return array.
targets_return	Returns an array of rendering target information.

Returns

Zero if unsuccessful; non-zero otherwise.

Description

This function returns information about which drawable types the server supports.

A drawable is specified only to indicate the screen for which the returned information should apply. None of the other drawable attributes are used.

The depth value is specified to indicate the depth for which the returned information should apply. If the application wants information on all supported depths, a value of zero should be specified.

The drawable type is specified to indicate the type of drawable for which the returned information should apply. The type field of the **PEXRenderingTarget** data structure will have these same values with the exception of **PEXAnyDrawable**.

The visual is specified to indicate the visual for which the returned information should apply. If the application wants information on all supported visuals, a null pointer should be specified.

PEXlib allocates memory for the returned target values. **XFree** should be called to deallocate the memory.

Data Structures

```
typedef struct {
    int                 depth;
    int                 type;/* PEXWindowDrawable, PEXPixmapDrawable, PEXBufferDrawa
    Visual              *visual;
} PEXRenderingTarget;
```

Errors

BadDrawable	The specified drawable resource identifier is invalid.
BadValue	The specified visual is invalid.

See Also

PEXGetProtocolFloatFormat - Return Float Format used on Specified Display Connection

Synopsis

int PEXGetProtocolFloatFormat(Display **display*)

Arguments

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

Returns

Protocol floating point format (see **PEXGetEnumTypeInfo**); zero if unsuccessful.

Description

This function returns the protocol floating point format being used by PEXlib on the specified display connection.

Errors

None

See Also

PEXGetSizeOCs - Return the Protocol Formatted Size of Output Commands

Synopsis

int PEXGetSizeOCs(int *float_format*, int *oc_count*, PEXOCData **oc_data*)

Arguments

float_format	The floating point format to use in computing the formatted size.
oc_count	The number of output commands.
oc_data	An array of 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 output commands. An unsuccessful return value is possible if the specified floating point format is invalid or unsupported by PEXlib.

Data Structures

```
See PEXlib.h for the definition of PEXOCData.
```

Errors

None

See Also

Chapter 2

Output Commands

The functions described in this chapter direct output commands to the specified display. Output commands may be rendered immediately or stored in a structure resource. The first three parameters of all output command functions are identical: the first parameter specifies the display connection, the second parameter specifies the resource identifier for the targeted renderer or structure, and the third parameter specifies whether the output commands are to be rendered immediately (in which case the second parameter identifies the renderer resource) or stored in a structure (in which case the second parameter identifies the structure resource). In order for a PHIGS workstation resource to process output commands for rendering, the output commands must be stored in structures. See *Chapter 10: PHIGS Workstations*.

Note, above, that the third parameter to each output command function (the request type parameter) determines how PEXlib will send the output command to the PEX server. This parameter must have one of four values: **PEXOCRender**, **PEXOCRenderSingle**, **PEXOCStore** or **PEXOCStoreSingle**. The first two, **PEXOCRender** and **PEXOCRenderSingle**, will cause the output command to be formatted into a **PEXRCRenderOutputCommands** request. This request will cause the server to immediately render the output commands using the specified renderer when the request is received by the server. **PEXOCStore** and **PEXOCStoreSingle** will cause the output command to be formatted into a **PEXRCStoreElements** request. This request will cause the server to store the output commands into the specified structure when the request is received by the server.

In order to improve performance, PEXlib will try to pack output commands into single **PEXRCRenderOutputCommands** or **PEXRCStoreElements** requests. This packing, or buffering, of output commands will normally occur when the request type is **PEXOCRender** or **PEXOCStore**. When the request type is **PEXOCRenderSingle** or **PEXOCStoreSingle**, PEXlib will force the output command to be formatted into a single request by itself. This is useful, as an example, for debugging purposes.

Also, behavior of the **PEXRCStoreElements** request when the specified structure's editing mode is **PEXStructureReplace** is not deterministic when using the request type **PEXOCStore**. The **PEXRCStoreElements** request will replace a range of elements in the structure dependent upon how many output commands are in the request. The number of output commands in a given request is dependent upon too many variables to predict. Applications should either use **PEXOCStoreSingle** when the structure's editing mode is **PEXStructureReplace**, or delete elements and use **PEXStructureInsert** to insert the new elements. See *Chapter 7: Structures* for more detail.

Finally, PEXlib will not pack a single request so that it is too large to send to the server. However, if a single OC is too large to send, PEXlib will cause the server to produce a **BadLength** error. The application can determine before-hand if an OC is too large by calling **PEXGetSizeOCs** and **XMaxRequestSize**. See *Chapter 1: Introduction to PEXlib* for more detail.

2.1. Primitives and Attributes

Output commands processed by a renderer or PHIGS workstation to produce visible output (that is, pixels on the display screen) are called output primitives or simply primitives. PEX supports a number of output primitives, including markers, text and annotation text, polylines and non-uniform B-spline curve, fill areas, fill area sets, triangle strips, quadrilateral meshes, non-uniform B-spline surfaces, cell arrays and a generalized drawing primitive (or GDP).

Output commands which alter the rendering pipeline state are called output primitive attributes or simply attributes. The initial renderer or PHIGS workstation pipeline state is provided by a pipeline context resource or defaults if no pipeline context resource was specified in the creation of the renderer or PHIGS workstation resource. After the renderer or PHIGS workstation has obtained its initial pipeline state, pipeline attributes are modifiable only by the output primitive attribute functions described in this chapter. It is not possible to directly query the current pipeline state.

More detailed information on attributes is given in *Chapter 4: Pipeline Contexts*.

Attributes that affect all output primitives are:

HLHSR identifier
color approximation index
depth cue index
global transformation
local transformation
model clip flag
model clip volume
name set
pick id
rendering color model
view index

2.1.1. Markers

A marker is a two-dimensional glyph, or symbol, representing a location in three-dimensional space. Modeling coordinates specify the marker's position. The position is transformed within the rendering pipeline like other output primitive coordinates. Since a marker has no surface from which to reflect light, the light-source illumination stage of the rendering pipeline has no effect on markers. When depth-cueing is enabled, a marker's color is affected by the depth-cueing computation.

The attributes that affect marker primitives are:

marker bundle index
marker color
marker color asf
marker scale
marker scale asf
marker type
marker type asf

The marker color is the intrinsic color of the marker. This color is affected by the depth-cueing computation, if depth-cueing is enabled. The resulting color is then transformed, by the color approximation stage of the rendering pipeline, into a color the device can display. There are two output commands for setting the marker color attribute. The first allows an application to set the marker color to a particular indexed color value. The second is more general, allowing the marker color to be set to an indexed color value or a direct color value.

The marker type indicates the glyph used to represent the marker's position. Supported values for marker type are inquirable via **PEXGetEnumTypeInfo**.

The marker scale is used to control the size of the rendered marker glyph. The nominal size of a marker is the size used if the marker scale is set to 1.0. (See **PEXGetImpDepConstants**.) The marker's displayed size is computed by multiplying the marker's nominal size by the marker scale value, and then selecting the nearest available size for the marker. An exception to this is **PEXMarkerDot** which is always displayed as the smallest dot the workstation can display.

Whenever a marker primitive is rendered, each of the marker attributes is obtained directly from the current pipeline state or indirectly from a marker bundle table. The source for the attribute value is determined by the current settings of the marker aspect source flags, or ASFs. If an attribute's ASF is set to **PEXIndividual**, the attribute value is obtained directly from the current pipeline state. If an attribute's ASF is set to **PEXBundled**, the attribute value is obtained from the current marker bundle table. The marker bundle index attribute indicates the marker bundle table entry to be used. Marker color asf, marker type asf, and marker scale asf are the attributes which control the source of marker attribute values.

The treatment of markers during hidden surface computations depends on the HLHSR (Hidden Line/Hidden Surface Renderer) mode selected and its implementation. Z-buffer algorithms usually perform the hidden surface computation on the realization of marker glyphs. Scan-line algorithms may determine visibility for markers solely from the 3D position of the marker. Clipping is performed on the marker position. If the marker position is outside the clipping volume, no portion of the marker glyph will be rendered. Clipping of portions of marker glyphs that extend beyond the clipping volume is implementation-dependent.

2.1.2. Text

Two types of text primitives are supported: regular text and annotation text. Regular text primitives are rendered in an arbitrary 3D plane, while annotation text appears parallel to the view surface.

A text string is a list of characters rendered at a particular location in three-dimensional space. Text positions are specified in modeling coordinates and transformed into screen coordinates within the rendering pipeline. A text string has no surface from which to reflect light, so the light-source illumination stage of the rendering pipeline has no effect. Depth-cueing, if enabled, does affect the color of text primitives.

The attributes that affect regular text primitives are:

character expansion
character expansion asf
character height
character spacing
character spacing asf
character up vector
text alignment
text bundle index
text color
text color asf
text font index
text font index asf
text path
text precision
text precision asf

The text color is the intrinsic color of the text or annotation text string. This color is affected by the depth-cueing computation, if depth-cueing is enabled. The resulting color is transformed, by the color approximation stage of the rendering pipeline, into a color the device can display. There are two output commands for setting the text color attribute. The first allows an application to set the text color to a particular indexed color value. The second is more general, allowing the text color to be set to either an indexed color value or a direct color value.

The text font index specifies which entry in the current font table (i.e. which font group) is to be used. Character glyph definitions are obtained from the font resource identifiers stored in the font table entry indicated by the text font index.

The remaining attributes are defined exactly as they are for PHIGS. Examples of the functionality and use of each is found in the PHIGS specification document. (See PHIGS functional specification, ISO/IEC 9592-1:1988, section 4.5.5)

When a text or annotation text primitive is rendered, attributes are obtained directly from the current pipeline state or indirectly from a text bundle table. The source for the attribute value is determined by the current settings of the corresponding ASFs. If an attribute's ASF is set to **PEXIndividual**, the attribute value is obtained directly from the current pipeline state. If an attribute's ASF is set to **PEXBundled**, the attribute value is obtained from the current text bundle table. The text bundle table entry used is indicated by the text bundle index attribute. Text color asf, text font index asf, text precision asf, character expansion asf, and character spacing asf are attributes which control the source of text attribute values.

The treatment of text and annotation text strings during hidden surface computations depends on the HLHSR mode selected and its implementation. Z-buffer algorithms usually perform the hidden surface computation on the 2D realization of the text string. Scan-line algorithms may determine visibility for text strings only from the 3D positions of each string. Clipping of text strings is dependent on the text precision attribute. If the text precision is **PEXStringPrecision**, clipping is done in an implementation-dependent fashion. If the text precision is **PEXCharPrecision**, clipping is done on at least a character-by-character basis. If the text precision is **PEXStrokePrecision**, each character is clipped to the clipping boundaries.

2.1.3. Annotation Text

Annotation text primitives and text primitives are the same except that annotation primitives must appear parallel to the viewing surface. Some attributes are shared with text primitives, and other text attributes are duplicated to provide independent control over annotation text appearance. The annotation text style attribute specifies whether a connecting line is rendered from the reference point to the annotation text string. Supported values for annotation text style are inquirable via **PEXGetEnumTypeInfo**. If the style is **PEXATextConnected**, the connecting line is rendered with the current polyline attributes.

The attributes that affect annotation text primitives are:

annotation text alignment
annotation text height
annotation text path
annotation text style
annotation text up vector
character expansion
character expansion asf
character spacing
character spacing asf
text bundle index
text color
text color asf
text font index
text font index asf
text precision
text precision asf

2.1.4. Polylines

A polyline is specified as a list of vertices connected with line segments.

The attributes that affect polyline primitives are:

line bundle index
line color
line color asf
line type
line type asf
line width
line width asf
polyline interpolation
polyline interpolation asf

The line color is the intrinsic color for line primitives without vertex colors. This color is affected by the depth-cueing computation, if depth-cueing is enabled. The resulting color is then transformed, by the color approximation stage of the rendering pipeline, into a color the device can display. There are two output commands for setting the line color attribute. The first allows an application to set the line color to a particular indexed color value. The second is more general, allowing the line color to be set to either an indexed color value or a direct color value.

The line type indicates the visual style of line primitives rendered on the screen. Supported values for line type are inquirable via **PEXGetEnumTypeInfo**. For lines types other than **PEXLineTypeSolid**, the length of dots, dashes, and spaces is implementation-dependent. The line width attribute is a scale factor that is multiplied by the nominal line width for the display. (See **PEXGetImpDepConstants**.) The line type and line width attributes are applied to the 2D projection of the line in device coordinates, not to the 3D primitive itself.

Polyline set primitives may include a color value at each vertex. When vertex colors are provided, they take precedence over the line color attribute. The polyline interpolation method controls how color values provided at the vertices are used when rendering polylines. If the polyline interpolation method is **PEXPolylineInterpNone** and colors are provided at the vertices, the color of the ith vertex will be used to render the line between the ith and the i+1th vertices (leaving the color for last vertex unused) rendering each polyline segment in a constant color. If the interpolation method is **PEXPolylineInterpColor**, a linear interpolation is performed between the vertex colors to render points along each line segment. The interpolated colors are then depth-cued and transformed into device colors.

When a line primitive is rendered, each attribute is obtained directly from the current pipeline state, or indirectly from a line bundle table. The source for the attribute value is determined by the current settings of the corresponding ASFs. If an attribute's ASF is set to **PEXIndividual**, the attribute value is obtained directly from the current pipeline state. If an attribute's ASF is set to **PEXBundled**, the attribute value is obtained from the current line bundle table. The line bundle table entry to be used is indicated by the line bundle index attribute. Line color asf, line type asf, line width asf, and polyline interpolation asf are attributes which control source of polyline attribute values.

2.1.5. Non-Uniform Rational B-Spline Curves

Non-uniform B-spline curves are defined as a list of knot values and a set of control points.

The attributes that affect non-uniform B-spline curve primitives are:

curve approximation
curve approximation asf
line bundle index
line color
line color asf
line type
line type asf
line width
line width asf

Non-uniform B-spline curves use all of the attributes of polylines, except for the polyline interpolation method. These curves also use the curve approximation attribute to define the fidelity with which non-uniform B-spline curves are rendered. Supported values for curve approximation are inquirable via **PEXGetEnumTypeInfo**.

2.1.6. Fill Areas

Fill areas are a type of surface primitive. A surface extending beyond the clipping boundary is clipped, and only the portion within the clipping boundary is rendered. Surfaces are the only output primitives affected by the reflectance calculations. Surface primitives include fill area (with and without data), fill area set (with and without data), set of fill area sets, triangle strip, quadrilateral mesh, and non-uniform B-spline surface.

A fill area is a sequence of points defining the boundary of a planar surface. The last point is always implicitly connected to the first point so the area is closed. Fill areas are not required to be strictly planar, but shading artifacts may occur if the fill area is not planar or nearly so. The odd-even rule is used for determining the area that lies in the interior of the fill area.

Part of the definition of a fill area is the shape specification. This parameter is used to indicate whether the shape of the fill area is **PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex** or **PEXShapeUnknown**. A shape of **PEXShapeComplex** means the fill area may have self-intersecting edges. A shape of **PEXShapeNonConvex** means the fill area has no self-intersecting edges but some of the interior angles are not convex. A shape of **PEXShapeConvex** means all of the fill area's interior angles are convex. A shape of **PEXShapeUnknown** means the shape of the fill area is not known. The server will have to determine its shape or assume it is **PEXShapeComplex**.

The attributes that affect fill area (all surface) primitives are:

back-face surface color
back-face surface color asf
back-face surface interior style
back-face surface interior style asf
back-face surface interior style index
back-face surface interior style index asf
back-face surface interpolation method
back-face surface interpolation method asf
back-face surface reflection attributes
back-face surface reflection attributes asf
back-face surface reflection model
back-face surface reflection model asf
facet culling mode
facet distinguish flag
interior bundle index
light state
pattern reference point
pattern reference vectors
pattern size
surface color
surface color asf
surface interior style
surface interior style asf
surface interior style index
surface interior style index asf
surface interpolation method
surface interpolation method asf
surface reflection attributes
surface reflection attributes asf
surface reflection model
surface reflection model asf
surface edge color
surface edge color asf
surface edge type
surface edge type asf
surface edge width
surface edge width asf
surface edge bundle index
surface edges
surface edges asf

If the facet distinguish flag is enabled then all back-facing surfaces will be rendered using the back-facing attribute values. The culling mode must be set to **PEXNone**, **PEXBackFaces** or **PEXFrontFaces**. This determines whether no surfaces are culled, back-facing surfaces are culled or front-facing surfaces are culled. The facet normal, either the one supplied or, if not supplied, the one computed, for each facet is used to determine whether the facet is front-facing or back-facing.

The surface color is affected by the reflectance calculations (which utilizes the interior style, the reflection model and reflection attributes) and the depth-cueing computations. The resulting color is then transformed, by the color approximation stage of the rendering pipeline, into a color the device can display.

The interior style determines how the interior of the fill area is rendered. Supported values for surface interior style are inquirable via **PEXGetEnumTypeInfo**. An interior style of **PEXInteriorStyleEmpty** means the interior is not rendered at all. An interior style of **PEXInteriorStyleSolid** means the interior is solid filled. An interior style of **PEXInteriorStyleHollow** means only the boundary of the interior area is rendered, and if the fill area is clipped, the boundary along the clipped edge is also rendered. An interior style of **PEXInteriorStylePattern** means the interior is pattern filled using the pattern attributes (pattern reference point, pattern reference vectors and pattern size). An interior style of **PEXInteriorStyleHatch** means the interior is hatch filled.

The light state specifies indices into the light table of those entries (or lights) which are currently enabled. If the light table contains defined entries that are not referenced by the light state attribute, those lights are disabled.

The reflection model specifies the method that is used to perform the reflectance calculations when rendering surface primitives. Supported values for reflection model are inquirable via **PEXGetEnumTypeInfo**. If the reflection model is **PEXReflectionNone** then no reflectance calculation is performed. If the reflection model is **PEXReflectionAmbient** then only the ambient terms of the lighting equation are used. If the reflection model is **PEXReflectionDiffuse** then only the ambient and diffuse terms of the lighting equation are used. If the reflection model is **PEXReflectionSpecular** then the ambient, diffuse and specular terms of the lighting equation are all used during the reflectance calculations.

The reflection attributes indicate how a primitive reflects light. These include ambient, diffuse and specular coefficients, the specular concentration and color, and the transmission coefficient (also thought of as a transparency coefficient).

The surface interpolation method controls how shading is performed when rendering surface primitives. Supported values for surface interpolation method are inquirable via **PEXGetEnumTypeInfo**. If the surface interpolation method is **PEXSurfaceInterpNone** then the color resulting from a single light source computation is used for the entire surface. If the surface interpolation method is **PEXSurfaceInterpColor** then the colors are computed at the vertices of the surface according to the current reflection model. These color values are then linearly interpolated across the interior of the surface or the edges. If the surface interpolation method is **PEXSurfaceInterpDotProduct** then the lighting equation dot products are computed at the vertices. These dot products are linearly interpolated and the reflectance calculation is applied using these values to compute the color value at each pixel in the interior of a surface or along a surface edge. If the surface interpolation method is **PEXSurfaceInterpNormal** then an attempt is made to interpolate the normal across the facet and perform the reflectance calculation as accurately as possible at each pixel in the interior of a surface or along a surface edge.

When a surface is rendered, each attribute is obtained directly from the current pipeline state, or indirectly from the interior bundle table. The source for the attribute value is determined by the current settings of the corresponding ASFs. If an attribute's ASF is set to **PEXIndividual**, the attribute value is obtained directly from the current pipeline state. If an attribute's ASF is set to **PEXBundled**, the attribute value is obtained from the current interior bundle table. The interior bundle table entry to be used is indicated by the interior bundle index attribute. Surface interior style asf, surface interior style index asf, surface color asf, surface reflection attributes asf, surface reflection model asf, surface interpolation method asf, back-face surface interior style asf, back-face surface interior style index asf, back-face surface surface color asf, back-face surface reflection attributes asf, back-face surface reflection model asf and back-face surface interpolation method asf are attributes which control the source of fill area attribute values.

The surface edges attribute indicates whether fill area edges are rendered in addition to fill area interiors. Surface edges for fill areas are the edges of each fill area. If surface edges are rendered, the surface edge color, surface edge type, and surface edge width are used to render them. These attributes are analogous to the attributes that affect line primitives. Surface edge colors are depth-cued but are not affected by light-source computations. Surface edge type and width, like line type and width, are raster attributes applied after the surface edge has been transformed into device coordinates. Edges are considered to be rendered "on top of" fill area interiors. This means that surface edges, when enabled, should be visible no matter what interior style is used. Parts of clipped surface edges are not visible (no new edge is generated by the clipping process). Also the source of the edge attribute value is determined by the surface edge color asf, surface edge type asf, and surface edge width asf.

Some fill area primitives allow additional data for the facet and for each vertex. If a color value is specified for the facet, it is taken to be the intrinsic color of the surface and is used instead of the current surface color when the fill area is rendered. Vertex colors take precedence over facet colors which override the surface color attributes. A normal specified for the fill area facet is assumed to be a surface normal for the facet and is used for back-facing surface processing. This normal may also be used in reflectance calculations. If no fill area facet normal is provided, one is computed by determining the first three non-colinear vertices A, B, and C, and computing the cross product of the vector from A to B with the vector from A to C. If the fill area is degenerate (for example, does not contain three non-colinear vertices), the resultant value of the normal is implementation-dependent.

2.1.7. Fill Area Sets

A fill area set consists of multiple fill areas, each of which defines a "contour", "island" or a "hole" that is part of a single, planar, area-defining output primitive. All of the fill area attributes described in the section on fill areas apply to fill area sets. If a normal is not provided for the fill area set then it is computed, using the method described in the section on fill areas, for the first contour.

Part of the definition of a fill area set is the contour hint flag. This parameter is used to indicate whether the contours are **PEXContourDisjoint**, **PEXContourNested**, **PEXContourIntersecting** or **PEXContourUnknown**. A contour hint of **PEXContourDisjoint** means all contours are spatially disjoint. No overlapping or intersection occurs between any contours in the fill area set. A contour hint of **PEXContourNested** means the contours are either disjoint or wholly contained within another contour. No contour will have edges that intersect or are coincident with edges of any other contour. A contour hint of **PEXContourIntersecting** means that separate contours may have edges that are coincident or overlap. A contour hint of **PEXContourUnknown** means nothing is known about the relationships between the contours. Fill area sets with contours that have higher complexity interrelationships than that indicated by their contour hint parameter are rendered in an implementation-dependent manner.

A set of fill area sets may be connected where individual fill area sets may share geometry and attribute information at vertices. Each fill area set defines a planar surface, but each fill area set is not necessarily co-planar with the other fill area sets. One benefit to the set of fill area sets primitive is that data is transmitted across the network once per unique vertex instead of once for every fill area set sharing the vertex. An additional parameter specifies whether the contours are all 1, i.e. there is only one contour in each fill area set. If **True**, then each fill area set in the SOFAS consists of a single facet, or fill area.

2.1.8. Triangle Strips

A triangle strip is specified as a list of n vertices that represent n-2 triangles. For each value of i less than or equal to n-2, the ith, i+1th, and i+2th vertices define a triangle. Geometric normals are computed in the same way as described for fill areas, except that the order of vertices is reversed for even-numbered triangles in the triangle strip. The set of triangles need not be co-planar.

All of the fill area attributes described in the section on fill areas, except for the shape flag, affect triangle strips. The surface edges for triangle strips are the line segments forming the boundaries of all triangles in the strip. There is only one edge between any two adjacent triangles.

2.1.9. Quadrilateral Meshes

A quadrilateral mesh is specified as an array of vertices representing quadrilaterals. For each i less than the number of rows minus one and each j less than the number of columns minus one, a quadrilateral is formed by the vertices (i,j), (i+1,j), (i+1,j+1) and (i,j+1). Normals are assumed to be unit length vectors. Normals for quadrilaterals, if not provided explicitly, are computed by taking the cross product of the diagonals. For a quadrilateral with the above vertices, the cross product would be formed as follows:

normal(i,j) = (point(i+1, j+1) - point(i, j)) X (point(i, j+1) - point(i+1, j))

The set of quadrilaterals need not be co-planar, however, each individual quadrilateral is assumed to be planar.

All of the fill area attributes described in the section on fill areas affect quadrilateral meshes. The surface edges for quadrilateral meshes are the line segments forming the boundaries of the individual quadrilaterals in the mesh. There is only one edge between any two adjacent quadrilaterals.

2.1.10. Non-Uniform Rational B-Spline Surfaces

A non-uniform B-spline surface is a B-spline surface specified by two lists of knot values and a set of control points. A set of trimming loops may be specified along with the data for a non-uniform B-spline surface to specify those areas of the surface to be rendered.

Non-uniform rational B-spline surfaces are affected by the following list of attributes in addition to the fill area attributes described in the section on fill areas:

parametric surface characteristics
surface approximation
surface approximation asf

The surface approximation defines the approximation method and accuracy with which non-uniform B-spline surfaces are rendered. Supported values for surface approximation are inquirable via **PEXGetEnumTypeInfo**.

The parametric surface characteristics defines the surface characteristics. Supported values for parametric surface characteristics are inquirable via **PEXGetEnumTypeInfo**.

2.1.11. Cell Arrays

A cell array is a parallelogram of equal-sized cells, each of which is a parallelogram and has a single color. The attributes affecting cell arrays are those listed in *2.1: Primitives and Attributes*.

2.1.12. Generalized Drawing Primitives

A generalized drawing primitive (GDP) output command is provided. This allows individual implementations to extend the supported primitive set. There are no standard PEX GDPs. The complete interface and behavior for each specified GDP identifier should be available with the individual PEX implementations.

2.2. Output Primitive Functions

This section describes all the output primitive functions.

2.2.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* coordinates */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXCoord;

      typedef struct {
          float               x;
          float               y;
      } PEXCoord2D;

      typedef struct {
          unsigned long       count;                  /* number of points */
          PEXCoord            *points;
      } PEXListOfCoord;       /* Pointer to an array of 3D points */

      typedef struct {
          unsigned long       count;                  /* number of points */
          PEXCoord2D          *points;
      } PEXListOfCoord2D;     /* Pointer to an array of 2D points */

/* vectors */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXVector;

/* facet data */

      typedef union {
          PEXColorIndexed     index;
          PEXColorRGB         rgb;
          PEXColorHSV         hsv;
          PEXColorHLS         hls;
          PEXColorCIE         cie;
          PEXColorRGB8        rgb8;
          PEXColorRGB16       rgb16;
          PEXVector           normal;
          PEXColorIndexedNormal index_normal;
          PEXColorRGBNormal   rgb_normal;
          PEXColorHSVNormal   hsv_normal;
          PEXColorHLSNormal   hls_normal;
          PEXColorCIENormal   cie_normal;
          PEXColorRGB8Normal  rgb8_normal;
          PEXColorRGB16Normal rgb16_normal;
      } PEXFacetData;
```

```
typedef union {
    PEXColorIndexed     *index;
    PEXColorRGB         *rgb;
    PEXColorHSV         *hsv;
    PEXColorHLS         *hls;
    PEXColorCIE         *cie;
    PEXColorRGB8        *rgb8;
    PEXColorRGB16       *rgb16;
    PEXVector           *normal;
    PEXColorIndexedNormal *index_normal;
    PEXColorRGBNormal   *rgb_normal;
    PEXColorCIENormal   *cie_normal;
    PEXColorHSVNormal   *hsv_normal;
    PEXColorHLSNormal   *hls_normal;
    PEXColorRGB8Normal  *rgb8_normal;
    PEXColorRGB16Normal *rgb16_normal;
} PEXArrayOfFacetData;

typedef struct {
    PEXTableIndex       index;
    unsigned short      reserved;
} PEXColorIndexed;

typedef unsigned short  PEXTableIndex;

typedef struct {
    float               red;
    float               green;
    float               blue;
} PEXColorRGB;

typedef struct {
    float               hue;
    float               saturation;
    float               value;
} PEXColorHSV;

typedef struct {
    float               hue;
    float               lightness;
    float               saturation;
} PEXColorHLS;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXColorCIE;

typedef struct {
    unsigned char       red;
    unsigned char       green;
    unsigned char       blue;
    unsigned char       reserved;
```

```
    } PEXColorRGB8;

    typedef struct {
        unsigned short      red;
        unsigned short      green;
        unsigned short      blue;
        unsigned short      reserved;
    } PEXColorRGB16;

    typedef struct {
        PEXColorIndexed     index;
        PEXVector           normal;
    } PEXColorIndexedNormal;

    typedef struct {
        PEXColorRGB         rgb;
        PEXVector           normal;
    } PEXColorRGBNormal;

    typedef struct {
        PEXColorCIE         cie;
        PEXVector           normal;
    } PEXColorCIENormal;

    typedef struct {
        PEXColorHSV         hsv;
        PEXVector           normal;
    } PEXColorHSVNormal;

    typedef struct {
        PEXColorHLS         hls;
        PEXVector           normal;
    } PEXColorHLSNormal;

    typedef struct {
        PEXColorRGB8        rgb8;
        PEXVector           normal;
    } PEXColorRGB8Normal;

    typedef struct {
        PEXColorRGB16       rgb16;
        PEXVector           normal;
    } PEXColorRGB16Normal;

/* vertex data */

    typedef struct {
        unsigned long       count;                  /* number of vertices */
        PEXArrayOfVertex    vertices;               /* pointer to vertices */
    } PEXListOfVertex;

    typedef union {
        PEXCoord                    *no_data;
        PEXVertexIndexed            *index;
        PEXVertexRGB                *rgb;
```

```
    PEXVertexHSV                *hsv;
    PEXVertexHLS                *hls;
    PEXVertexCIE                *cie;
    PEXVertexRGB8               *rgb8;
    PEXVertexRGB16              *rgb16;
    PEXVertexNormal             *normal;
    PEXVertexEdge               *edge;
    PEXVertexIndexedNormal      *index_normal;
    PEXVertexRGBNormal          *rgb_normal;
    PEXVertexHSVNormal          *hsv_normal;
    PEXVertexHLSNormal          *hls_normal;
    PEXVertexCIENormal          *cie_normal;
    PEXVertexRGB8Normal         *rgb8_normal;
    PEXVertexRGB16Normal        *rgb16_normal;
    PEXVertexIndexedEdge        *index_edge;
    PEXVertexRGBEdge            *rgb_edge;
    PEXVertexHSVEdge            *hsv_edge;
    PEXVertexHLSEdge            *hls_edge;
    PEXVertexCIEEdge            *cie_edge;
    PEXVertexRGB8Edge           *rgb8_edge;
    PEXVertexRGB16Edge          *rgb16_edge;
    PEXVertexNormalEdge         *normal_edge;
    PEXVertexIndexedNormalEdge  *index_normal_edge;
    PEXVertexRGBNormalEdge      *rgb_normal_edge;
    PEXVertexHSVNormalEdge      *hsv_normal_edge;
    PEXVertexHLSNormalEdge      *hls_normal_edge;
    PEXVertexCIENormalEdge      *cie_normal_edge;
    PEXVertexRGB8NormalEdge     *rgb8_normal_edge;
    PEXVertexRGB16NormalEdge    *rgb16_normal_edge;
} PEXArrayOfVertex;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
} PEXVertexIndexed;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
} PEXVertexRGB;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
} PEXVertexHSV;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
} PEXVertexHLS;

typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
```

```
} PEXVertexCIE;

typedef struct {
    PEXCoord            point;
    PEXColorRGB8        rgb8;
} PEXVertexRGB8;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
} PEXVertexRGB16;

typedef struct {
    PEXCoord            point;
    PEXVector           normal;
} PEXVertexNormal;

typedef struct {
    PEXCoord            point;
    unsigned int        edge;
} PEXVertexEdge;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
    PEXVector           normal;
} PEXVertexIndexedNormal;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
    PEXVector           normal;
} PEXVertexRGBNormal;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
    PEXVector           normal;
} PEXVertexHSVNormal;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
    PEXVector           normal;
} PEXVertexHLSNormal;

typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
    PEXVector           normal;
} PEXVertexCIENormal;

typedef struct {
    PEXCoord            point;
```

```
    PEXColorRGB8        rgb8;
    PEXVector           normal;
} PEXVertexRGB8Normal;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
    PEXVector           normal;
} PEXVertexRGB16Normal;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
    unsigned int        edge;
} PEXVertexIndexedEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
    unsigned int        edge;
} PEXVertexRGBEdge;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
    unsigned int        edge;
} PEXVertexHSVEdge;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
    unsigned int        edge;
} PEXVertexHLSEdge;

typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
    unsigned int        edge;
} PEXVertexCIEEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB8        rgb8;
    unsigned int        edge;
} PEXVertexRGB8Edge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
    unsigned int        edge;
} PEXVertexRGB16Edge;

typedef struct {
    PEXCoord            point;
```

```
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexNormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorIndexed     index;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexIndexedNormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorRGB         rgb;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexRGBNormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorHSV         hsv;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexHSVNormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorHLS         hls;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexHLSNormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorCIE         cie;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexCIENormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorRGB8        rgb8;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexRGB8NormalEdge;

    typedef struct {
        PEXCoord            point;
        PEXColorRGB16       rgb16;
        PEXVector           normal;
        unsigned int        edge;
    } PEXVertexRGB16NormalEdge;

/* similar to XErrorEvent - use to access additional info in OC
error */
```

```
typedef struct {
    int             type;
    Display         *display;      /* Display the event was read from */
    XID             resourceid;    /* resource id of renderer or structure */
    unsigned long   serial;        /* serial number of failed request */
    unsigned char   error_code;    /* error code of failed request */
    unsigned char   request_code;  /* Major op-code of failed request */
    unsigned char   minor_code;    /* Minor op-code of failed request */
    unsigned short  op_code;       /* op-code of failed output command */
    unsigned short  count;         /* number of output commands successfully */
                                   /* executed before error */
} PEXOCErrorEvent;
```

PEXAnnotationText - 3D Annotation Text Primitive

Synopsis

void PEXAnnotationText(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **origin*, PEXCoord **offset*, int *length*, char **string*)

Arguments

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**).
origin	The origin of the text string.
offset	The relative position of the text string from the origin.
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None

Description

This function creates an annotation text output primitive.

The first character set in the text font will be used.

The origin defines, in modeling coordinates, the position of the first character in the text string. The offset specifies an offset from the origin in normalized projection coordinates. The point where the text string is placed is called the annotation point. The annotation point is computed by adding the value of offset to the transformed origin point. The z-component of the annotation point specifies the x-y plane in normalized projection coordinate space on which the annotation text string is placed.

Depending on the text attribute source flag values, the text color, text precision, character expansion, character spacing, and text font index attributes are either obtained directly from the current text attribute values or from the entry in the text bundle specified by the current text bundle index attribute. The annotation text height, annotation text path, annotation text alignment, annotation text up vector, and annotation text style are also used to render the text string.

The annotation text string's color is affected only by the depth-cueing computation (the reflectance stage of the rendering pipeline affects only surface primitives) and is mapped to a device color. The entire annotation text primitive is clipped if the origin is clipped. If the origin is not clipped, the annotation text is clipped according to text clipping rules. The connection line, if any, is clipped according to polyline clipping rules, except that modeling clipping is ignored. The current set of polyline attributes is used to render the connection line.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetATextHeight**, **PEXSetATextUpVector**, **PEXSetATextPath**, **PEXSetATextAlignment**, **PEXSetATextStyle**, **PEXSetTextBundleIndex**

PEXAnnotationText2D - 2D Annotation Text Primitive

Synopsis

void PEXAnnotationText2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord2D **origin*, PEXCoord2D **offset*, int *length*, char **string*)

Arguments

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**).
origin	The origin of the text string.
offset	The relative position of the text string from the origin.
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None

Description

This function creates a 2D annotation text output primitive.

This primitive functions like **PEXAnnotationText** except that the origin and offset positions are specified using only x- and y-components, and the z-component is always assumed to be zero. This primitive is two-dimensional only in that the z-component is implied, since geometry transformations are still carried out in three dimensions.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetATextHeight**, **PEXSetATextUpVector**, **PEXSetATextPath**, **PEXSetATextAlignment**, **PEXSetATextStyle**, **PEXSetTextBundleIndex**

PEXCellArray - 3D Cell Array Primitive

Synopsis

void PEXCellArray(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **point1*, PEXCoord **point2*, PEXCoord **point3*, unsigned int *col_count*, unsigned int *row_count*, PEXTableIndex **color_indices*)

Arguments

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**).
point1	The first cell array definition point.
point2	The second cell array definition point.
point3	The third cell array definition point.
col_count	The number of cell columns (number of cells in the X direction).
row_count	The number of cell rows (number of cells in the Y direction).
color_indices	An array of color table index values which specify the color of each cell.

Returns

None

Description

This function creates a 3D cell-array output primitive.

A 3D cell-array primitive is a parallelogram of equally-sized cells, each of which is a parallelogram with a single color. Each cell has a width defined by:

$$width = \frac{\sqrt{(point1{\rightarrow}x - point2{\rightarrow}x)^2 + (point1{\rightarrow}y - point2{\rightarrow}y)^2 + (point1{\rightarrow}z - point2{\rightarrow}z)^2}}{col\_count}$$

and a height defined by:

$$height = \frac{\sqrt{(point1{\rightarrow}x - point3{\rightarrow}x)^2 + (point1{\rightarrow}y - point3{\rightarrow}y)^2 + (point1{\rightarrow}z - point3{\rightarrow}z)^2}}{row\_count}$$

Cell colors are specified in a one-dimensional array where the colors are stored in row-major order. The first color in the array is the color at the cell at the corner of point1, and subsequent colors represent the colors of cells proceeding to point2.

If any color index is not defined, color index one is used. If color index one is not defined, the resulting color is implementation-dependent. The column count and row count can not be zero.

Data Structures

```
typedef unsigned short  PEXTableIndex;
```

See also the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCellArray2D - 2D Cell Array Primitive

Synopsis

void PEXCellArray2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord2D **point1*, PEXCoord2D **point2*, unsigned int *col_count*, unsigned int *row_count*, PEXTableIndex **color_indices*)

Arguments

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**).
point1	The first cell array definition point.
point2	The second cell array definition point.
col_count	The number of cell columns (number of cells in the X direction).
row_count	The number of cell rows (number of cells in the Y direction).
color_indices	An array of color table index values which specify the color of each cell.

Returns

None

Description

This function creates a 2D cell array output primitive.

A 2D cell-array is a rectangle of equally-sized cells, each of which is a rectangle which has a single color. The primitive is specified by two points which define a rectangle aligned with the modeling coordinate axes. The width and height of each cell is defined as in **PEXCellArray** where the z component is zero.

All other aspects of this primitive are the same as **PEXCellArray**.

Data Structures

```
typedef unsigned short  PEXTableIndex;
```

See also the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXEncodedAnnoText - Encoded 3D Annotation Text Primitive

Synopsis

void PEXEncodedAnnoText(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **origin*, PEXCoord **offset*, unsigned int *count*, PEXEncodedTextData **encoded_text*)

Arguments

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**).
origin	The origin of the text string.
offset	The relative position of the text string from the origin.
count	The number of encoded text strings.
encoded_text	An array of encoded text strings.

Returns

None

Description

This function creates an annotation text output primitive.

This function is similar to **PEXAnnotationText**, except that multiple encoded text strings are specified. Each text string in the encoded text array has a character set, a character set width, an encoding state, and a list of characters.

The character set is an index into the current font group. Font groups contain individual fonts which are numbered starting at one; a value of three selects the third font in the font group currently being used. If a character set is not available in the current font group then the entire string will be rendered using the default font group. If a character set value is not available in the default font group then that portion of the string will be rendered in an implementation-dependent manner. The character set width indicates the width or size of characters in the strings. Valid values for character set width are **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The encoding state is provided for use by the application and is not interpreted by the PEX server.

All other aspects of this primitive are the same as **PEXAnnotationText**.

Data Structures

```
typedef struct {
    unsigned short      character_set;
    unsigned char       character_set_width;  /* PEXCSByte, PEXCSShort, PEXCSLong */
    unsigned char       encoding_state;
    unsigned short      reserved;
    unsigned short      length;
    char                *ch;
} PEXEncodedTextData;
```

See also the *Common Data Structures* section.

Errors

*BadPEXRenderer*The specified renderer resource identifier is invalid.

*BadPEXStructure*The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetATextHeight**, **PEXSetATextUpVector**, **PEXSetATextPath**, **PEXSetATextAlignment**, **PEXSetATextStyle**, **PEXSetTextBundleIndex**

PEXEncodedAnnoText2D - Encoded 2D Annotation Text Primitive

Synopsis

void PEXEncodedAnnoText2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord2D **origin*, PEXCoord2D **offset*, unsigned int *count*, PEXEncodedTextData **encoded_text*)

Arguments

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**).
origin	The origin of the text string.
offset	The relative position of the text string from the origin.
count	The number of encoded text strings.
encoded_text	An array of encoded text strings.

Returns

None

Description

This function creates a 2D annotation text output primitive.

This function is similar to **PEXAnnotationText2D**, except that multiple encoded text strings are specified. Each text string in the encoded text array has a character set, a character set width, an encoding state, and a list of characters.

The character set is an index into the current font group. Font groups contain individual fonts which are numbered starting at one; a value of three selects the third font in the font group currently being used. If a character set is not available in the current font group then the entire string will be rendered using the default font group. If a character set value is not available in the default font group then that portion of the string will be rendered in an implementation-dependent manner. The character set width indicates the width or size of characters in the strings. Valid values for character set width are **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The encoding state is provided for use by the application and is not interpreted by the PEX server.

All other aspects of this primitive are the same as **PEXAnnotationText2D**.

Data Structures

```
typedef struct {
    unsigned short      character_set;
    unsigned char       character_set_width;  /* PEXCSByte, PEXCSShort, PEXCSLong */
    unsigned char       encoding_state;
    unsigned short      reserved;
    unsigned short      length;
    char                *ch;
} PEXEncodedTextData;
```

See also the *Common Data Structures* section.

Errors

*BadPEXRenderer*The specified renderer resource identifier is invalid.

*BadPEXStructure*The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetATextHeight**, **PEXSetATextUpVector**, **PEXSetATextPath**, **PEXSetATextAlignment**, **PEXSetATextStyle**, **PEXSetTextBundleIndex**

PEXEncodedText - Encoded 3D Text Primitive

Synopsis

void PEXEncodedText(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **origin*, PEXVector **vector1*, PEXVector **vector2*, unsigned int *count*, PEXEncodedTextData **encoded_text*)

Arguments

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**).
origin	The origin of the text string.
vector1	A vector defining the positive x-direction of the text local coordinate system.
vector2	A vector defining the positive y-direction of the text local coordinate system.
count	The number of encoded text strings.
encoded_text	An array of encoded text strings.

Returns

None

Description

This function creates a text output primitive.

This function is similar to **PEXText**, except that multiple encoded text strings are specified. Each text string in encoded text array has a character set, a character set width, an encoding state, and a list of characters.

The character set is an index into the current font group. Font groups contain individual fonts which are numbered starting at one; a value of three selects the third font in the font group currently being used. If a character set is not available in the current font group then the entire string will be rendered using the default font group. If a character set value is not available in the default font group then that portion of the string will be rendered in an implementation-dependent manner. The character set width indicates the width or size of characters in the strings. Valid values for character set width are **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The encoding state is provided for use by the application and is not interpreted by the PEX server.

All other aspects of this primitive are the same as **PEXText**.

Data Structures

```
typedef struct {
    unsigned short      character_set;
    unsigned char       character_set_width;  /* PEXCSByte, PEXCSShort, PEXCSLong */
    unsigned char       encoding_state;
    unsigned short      reserved;
    unsigned short      length;
    char                *ch;
} PEXEncodedTextData;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetCharHeight**, **PEXSetCharUpVector**, **PEXSetTextPath**, **PEXSetTextAlignment**, **PEXSetTextBundleIndex**

PEXEncodedText2D - Encoded 2D Text Primitive

Synopsis

void PEXEncodedText2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord2D **origin*, unsigned int *count*, PEXEncodedTextData **encoded_text*)

Arguments

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**).
origin	The origin of the text string.
count	The number of encoded text strings.
encoded_text	An array of encoded text strings.

Returns

None

Description

This function creates a 2D text output primitive.

This function is similar to **PEXText2D**, except that multiple encoded text strings are specified. Each text string in encoded text array has a character set, a character set width, an encoding state, and a list of characters.

The character set is an index into the current font group. Font groups contain individual fonts which are numbered starting at one; a value of three selects the third font in the font group currently being used. If a character set is not available in the current font group then the entire string will be rendered using the default font group. If a character set value is not available in the default font group then that portion of the string will be rendered in an implementation-dependent manner. The character set width indicates the width or size of characters in the strings. Valid values for character set width are **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The encoding state is provided for use by the application and is not interpreted by the PEX server.

All other aspects of this primitive are the same as **PEXText2D**.

Data Structures

```
typedef struct {
    unsigned short      character_set;
    unsigned char       character_set_width;  /* PEXCSByte, PEXCSShort, PEXCSLong */
    unsigned char       encoding_state;
    unsigned short      reserved;
    unsigned short      length;
    char                *ch;
} PEXEncodedTextData;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**,
PEXSetCharExpansion, **PEXSetCharSpacing**, **PEXSetTextColorIndex**,
PEXSetTextColor, **PEXSetCharHeight**, **PEXSetCharUpVector**,
PEXSetTextPath, **PEXSetTextAlignment**, **PEXSetTextBundleIndex**

PEXExtendedCellArray - Extended 3D Cell Array Primitive

Synopsis

void PEXExtendedCellArray(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **point1*, PEXCoord **point2*, PEXCoord **point3*, unsigned int *col_count*, unsigned int *row_count*, int *color_type*, PEXArrayOfColor *colors*)

Arguments

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**).
point1	The first cell array definition point.
point2	The second cell array definition point.
point3	The third cell array definition point.
col_count	The number of cell columns (number of cells in the X direction).
row_count	The number of cell rows (number of cells in the Y direction).
color_type	The type of color for the cell colors (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
colors	An array of colors specifying the color of each cell.

Returns

None

Description

This function creates a 3D cell array output primitive.

This function is similar to **PEXCellArray**, except the colors are passed as either indexed color values or direct color values, depending on the color type.

Data Structures

```
typedef union {
    PEXColorIndexed     *indexed;
    PEXColorRGB         *rgb;
    PEXColorHSV         *hsv;
    PEXColorHLS         *hls;
    PEXColorCIE         *cie;
    PEXColorRGB8        *rgb8;
    PEXColorRGB16       *rgb16;
} PEXArrayOfColor;
```

See also the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXFillArea - 3D Fill Area Primitive

Synopsis

void PEXFillArea(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *shape_hint*, int *ignore_edges*, unsigned int *count*, PEXCoord **points*)

Arguments

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 of the fill area (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**).
ignore_edges	A flag that determines if surface edges are rendered (**True** or **False**).
count	The number of points.
points	An array of points defining the fill area.

Returns

None

Description

This function creates a fill area output primitive.

The area is defined by the list of points joined together to form a planar surface. Fill areas are not strictly required to be planar, but shading artifacts may occur if a fill area is not planar or nearly so. The first vertex of the fill area is connected to the second, the second to the third, and so on. The last vertex is implicitly connected to the first.

During the rendering process, the fill area vertices are transformed to positions in device coordinates. The surface colors are affected by the reflectance calculations which uses the light state, interior style, and reflection model attributes. Surface colors are further affected by the depth-cueing computation and then mapped to device colors. Fill areas outside the currently-defined clipping volume are not displayed. Fill areas crossing the clipping volume are clipped, and only the portions inside the clipping volume are displayed.

A fill area with fewer than three vertices is considered degenerate. It is stored in a structure, but when rendered, the primitive is ignored and has no visual effect.

A fill area may cross over itself to create a complex shape. The odd-even rule is used for determining the area that lies in the interior of the fill area. The shape hint is provided to enable performance improvements for certain shapes. Fill areas that are of higher complexity than indicated by the shape hint are rendered in an implementation-dependent manner. Consequently, applications should pass **PEXShapeUnknown** as the shape unless they are certain the fill area's shape is one of the other three.

The ignore edges flag is a boolean value specifying whether surface edges are rendered. If the flag is **True**, no surface edges are rendered for the fill area. If the ignore edges flag is **False**, surface edges are rendered according to the surface edge attributes if the surface edge flag attribute is **PEXOn**. Depending on the setting of the surface edge ASF values, the surface edges, surface edge color, surface edge type, and surface edge width attributes are obtained from one of two sources. These attributes are obtained directly from the current surface edge attribute values or from the edge bundle lookup table entry specified by the current edge bundle index attribute depending on the setting of the surface edge ASF attribute.

Depending on the setting of the surface attribute ASF values, the surface color, interior style, interior style index, surface interpolation method, and reflection model attributes are obtained from one of two sources. These attributes are obtained directly from the current surface attributes values or from the interior bundle lookup table entry specified by the current interior bundle index attribute.

When a surface is rendered, the surface color and reflection attributes are used to compute the colors of the surface if it is front-facing with respect to the point of view and the current culling mode allows front-faces to be rendered. If the surface is back-facing, the current distinguish mode is **True**, and the current culling mode allows back-faces to be rendered, the corresponding back-facing attributes are used instead.

Regardless of the fill area orientation, if the interior style is **PEXInteriorStylePattern**, the pattern size, pattern reference point and pattern reference vectors are used to pattern the fill area.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXFillArea2D - 2D Fill Area Primitive

Synopsis

void PEXFillArea2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *shape_hint*, int *ignore_edges*, unsigned int *count*, PEXCoord2D **points*)

Arguments

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 of the fill area (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**).

ignore_edges A flag that determines if surface edges are rendered (**True** or **False**).

count The number of points.

points An array of points defining the fill area.

Returns

None

Description

This function creates a 2D fill area output primitive.

This function is like **PEXFillArea**, except that the vertices consist of only x- and y-components. The z-component is assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. Geometry transformations are still carried out in three dimensions.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXFillAreaWithData - 3D Fill Area Primitive With Additional Data

Synopsis

void PEXFillAreaWithData(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *shape_hint*, int *ignore_edges*, unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, PEXFacetData **facet_data*, unsigned int *count*, PEXArrayOfVertex *vertices*)

Arguments

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 of the fill area (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**).
ignore_edges	A flag that determines if surface edges are rendered (**True** or **False**).
facet_attributes	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
vertex_attributes	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
color_type	The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
facet_data	A pointer to facet data.
count	The number of vertices.
vertices	An array of vertices defining the fill area.

Returns

None

Description

This function creates a fill area output primitive.

This function is like **PEXFillArea** except that it allows additional information to be specified for the fill area and for each vertex. Color values passed must be of the specified color type.

The facet attributes indicate the content of the facet data. This data may be a color, a normal, or a color followed by a normal. Use the constants **PEXGANone**, **PEXGAColor** and **PEXGANormal** to construct a mask indicating the data provided. If specified, the facet color takes precedence over the surface color. If specified, the facet normal is used to determine whether the fill area is back-facing.

The vertex attributes indicate the content of each fill area vertex. In addition to the coordinate (x,y,z), applications may specify a color, a normal, or a color followed by a normal for each vertex. Use the constants **PEXGANone**, **PEXGAColor** and **PEXGANormal** to construct a mask indicating the data provided. If specified, vertex colors will override facet color or the current surface color. If specified, vertex normals are taken to be normals at the vertices of the fill area.

The reflection model and the surface interpolation will affect how the additional data is used in rendering the surface.

Normals are assumed to be unit length vectors. The effect if the normal is not unit length is implementation-dependent.

All other aspects of this primitive are the same as **PEXFillArea**.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXFillAreaSet - 3D Set of Fill Areas Primitive

Synopsis

void PEXFillAreaSet(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *shape_hint*, int *ignore_edges*, int *contour_hint*, unsigned int *count*, PEXListOfCoord **point_lists*)

Arguments

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**).
count	The number of fill areas in the set.
point_lists	A pointer to the list of point arrays defining each contour of the fill area set.

Returns

None

Description

This function creates a fill area set output primitive.

This function is similar to **PEXFillArea**, but allows for the creation of areas with "islands" or "holes".

If any fill area in the set has fewer than three vertices, or if there are no contours defined, the primitive is considered degenerate. The primitive is stored in a structure, but when rendered, the primitive is ignored and has no visual effect.

A fill area set consists of an array of fill areas that define "contours" (disjoint pieces or holes) making up the primitive. Each fill area, or contour, is defined by a list of vertices joined together to form a planar surface.

The contour hint provides further information about the relationships between contours in the fill area set. If the contour hint is **PEXContourDisjoint**, all contours will be spatially disjoint. No overlapping or intersection occurs between any contours in the fill area set. If the contour hint is **PEXContourNested**, 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 the contour hint is **PEXContourIntersecting**, separated contours may have edges that are coincident or overlap. If the contour hint is **PEXContourUnknown** nothing is known about the interrelationships between contours. Fill area sets with contours that have higher complexity interrelationships than that indicated by the contour hint are rendered in an implementation-dependent manner.

The ignore edges flag is applied to each of the fill areas in the set.

All other aspects of this primitive are the same as **PEXFillArea**.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXFillAreaSet2D - 2D Set of Fill Areas Primitive

Synopsis

void PEXFillAreaSet2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *shape_hint*, int *ignore_edges*, int *contour_hint*, unsigned int *count*, PEXListOfCoord2D **point_lists*)

Arguments

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**).
count	The number of fill areas in the set.
point_lists	A pointer to the list of point arrays defining each contour of the fill area set.

Returns

None

Description

This function creates a 2D fill area set output primitive.

This function is like **PEXFillAreaSet**, except that the vertices consist of only x- and y-components. The z-component is assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. Geometry transformations are still carried out in three dimensions.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

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

Synopsis

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

Arguments

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**).
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**).
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 facet data.
vertex_lists	A pointer to the list of vertex arrays defining each contour of the fill area set.

Returns

None

Description

This function creates a fill area set output primitive.

This function is like **PEXFillAreaSet** except that it allows additional information to be specified for each fill area and for each vertex. It is similar to **PEXFillAreaWithData** but allows for the creation of areas with "islands" or "holes". Color values passed must be of the specified color type.

The facet attributes indicate the content of the facet data. This data may be a color, a normal, or a color followed by a normal. Use the constants **PEXGANone**, **PEXGAColor** and **PEXGANormal** to construct a mask indicating the data provided. If specified, the facet color takes precedence over the surface color. If specified, the facet normal is used to determine whether the fill area is back-facing.

The vertex attributes indicate the content of each fill area vertex. In addition to the coordinate (x,y,z), applications may specify a color, a normal, an edge flag, or any combination of the three specified in the order given. Use the constants **PEXGANone**, **PEXGAColor**, **PEXGANormal** and **PEXGAEdges** to construct a mask indicating the data provided. If specified, vertex colors will override facet color or the current surface color. If specified, vertex normals are taken to be normals at the vertices of the fill area.

The reflection model and the surface interpolation will affect how the additional data is used in rendering the surface. Edge controls are used to indicate those edges rendered if the surface edges are enabled. The edge control for vertex i indicates whether or not to render the edge between vertex i and vertex i+1. Surface edges are always rendered with the surface edge color and are not affected by the facet or vertex colors.

Normals are assumed to be unit length vectors. The effect if the normal is not unit length is implementation-dependent.

All other aspects of this primitive are the same as **PEXFillAreaSet**.

Data Structures

`See the` *Common Data Structures* `section.`

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXGDP - 3D Generalized Drawing Primitive

Synopsis

void PEXGDP(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, long *gdp_id*, unsigned int *count*, PEXCoord **points*, unsigned long *length*, char **data*)

Arguments

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**).
gdp_id	The identifier of the GDP.
count	The number of points.
points	The points used by the GDP.
length	The length, in bytes, of the data.
data	Additional data used by the GDP.

Returns

None

Description

This function creates a generalized drawing primitive.

The complete interface and behavior for each GDP identifier should be available with the individual PEX server implementations. There are no standard PEX GDPs. If the specified GDP identifier is not supported, then the output command is ignored.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGDP2D - 2D Generalized Drawing Primitive

Synopsis

void PEXGDP2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, long *gdp_id*, unsigned int *count*, PEXCoord2D **points*, unsigned long *length*, char **data*)

Arguments

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**).
gdp_id	The identifier of the GDP.
count	The number of points.
points	The points used by the GDP.
length	The length, in bytes, of the data.
data	Additional data used by the GDP.

Returns

None

Description

This function creates a 2D generalized drawing primitive.

The complete interface and behavior for each GDP identifier should be available with the individual PEX server implementations. There are no standard PEX GDPs. If the specified GDP identifier is not supported, then the output command is ignored.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXMarkers - 3D Markers Primitive

Synopsis

void PEXMarkers(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *count*, PEXCoord **points*)

Arguments

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	The number of points.
points	An array of points specifying marker locations.

Returns

None

Description

This function creates a markers output primitive.

A marker is a geometric primitive with a position as its only geometric attribute. The position is specified in modeling coordinates.

Rendering transforms the marker's position to a position in device coordinates. A marker has no geometric size, so geometric transformations do not affect the displayed size of the marker. The marker's color is affected only by depth-cueing and mapped to a device color. The clipping of markers whose position is inside the clipping volume but whose rendering is outside the clipping volume is implementation-dependent.

Depending on the setting of the marker ASF attributes, the marker color, marker type, and marker scale attributes are obtained either directly from the current marker attributes or from the marker bundle.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetMarkerType, **PEXSetMarkerScale**, **PEXSetMarkerColorIndex**, **PEXSetMarkerColor**, **PEXSetMarkerBundleIndex**

PEXMarkers2D - 2D Markers Primitive

Synopsis

void PEXMarkers2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *count*, PEXCoord2D **points*)

Arguments

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	The number of points.
points	An array of points specifying marker locations.

Returns

None

Description

This function creates a 2D markers output primitive.

This function is like **PEXMarkers**, except that the vertices consist of only x- and y-components. The z-component is assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. Geometry transformations are still carried out in three dimensions.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetMarkerType, **PEXSetMarkerScale**, **PEXSetMarkerColorIndex**, **PEXSetMarkerColor**, **PEXSetMarkerBundleIndex**

PEXNURBCurve - Non-Uniform Rational B-spline Curve Primitive

Synopsis

void PEXNURBCurve(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *rationality*, int *order*, float **knots*, unsigned int *count*, PEXArrayOfCoord *points*, double *tmin*, double *tmax*)

Arguments

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**).
rationality	The type of B-spline curve, whether rational or non-rational (**PEXRational** or **PEXNonRational**).
order	The order of the polynomial expression.
knots	An array of floats specifying the B-spline curve knots.
count	The number of control points that define the curve.
points	An array of control points defining the B-spline curve.
tmin	The minimum parameter value at which to evaluate the curve.
tmax	The maximum parameter value at which to evaluate the curve.

Returns

None

Description

This function creates a non-uniform B-spline curve output primitive.

The curve order is specified as a positive number. **PEXGetImpDepConstants** can be called to determine the largest supported value for curve order. The spline shape is specified using a list of knots in the parametric coordinate space and a list of control points in modeling coordinates. The number of control points must be at least as large as the order. The number of knots is the sum of the curve order plus the number of control points. The sequence of knots must be non-decreasing, i.e. $t(0) <= t(1) <= ... <= t(k-1)$ where k is the number of knots.

If the rationality is **PEXRational**, the control points must be specified in homogeneous (4D) modeling coordinates. PEXlib assumes that 4D modeling coordinates (x,y,z) have already been multiplied by the homogeneous coordinate (w). If the rationality is **PEXNonRational**, the control points must be specified in non-homogeneous (3D) modeling coordinates.

Evaluation of the spline is restricted to a specific region in the parametric coordinate space. The parametric limits, tmin and tmax, specify the region in the parametric coordinate space over which the spline is to be evaluated (tmin must be less than tmax). The parametric bounds must also satisfy the restriction $tmin >= t(order)$, $tmax <= t(k+1-order)$.

Depending on the ASF attributes, line color, line type, line width and curve approximation attributes are obtained either directly from the current line attributes or from the line bundle.

If the specified curve order is not supported, the output primitive is stored in a structure, but when rendered, the primitive is ignored and has no visual effect.

Also, the curve order must not be less than one.

Data Structures

```
typedef union {
    PEXCoord2D          *point_2d;
    PEXCoord            *point;
    PEXCoord4D          *point_4d;
} PEXArrayOfCoord;      /* Pointer to array of points */
```

See also the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetLineType, **PEXSetLineWidth**, **PEXSetLineColorIndex**, **PEXSetLineColor**, **PEXSetLineBundleIndex**, **PEXSetCurveApprox**, **PEXGetImpDepConstants**

PEXNURBSurface - Non-Uniform Rational B-spline Surface Primitive

Synopsis

void PEXNURBSurface(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *rationality*, int *uorder*, int *vorder*, float **uknots*, float **vknots*, unsigned int *col_count*, unsigned int *row_count*, PEXArrayOfCoord *points*, unsigned int *curve_count*, PEXListOfTrimCurve **trim_curves*)

Arguments

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**).
rationality	The type of B-spline surface, whether rational or non-rational (**PEXRational** or **PEXNonRational**).
uorder	The order of the polynomial expression in the u direction.
vorder	The order of the polynomial expression in the v direction.
uknots	An array of floats specifying the B-spline curve knots in the u direction.
vknots	An array of floats specifying the B-spline curve knots in the v direction.
col_count	The number of columns in the points array (number of points in the u direction).
row_count	The number of rows in the points array (number of points in the v direction).
points	An array of points defining the B-spline surface.
curve_count	The number of trimming curves.
trim_curves	A pointer to a list of trimming curves.

Returns

None

Description

This function creates a non-uniform B-spline surface output primitive.

The surface is generated as a function of the parametric variables u and v. The u and v order must be positive integers and indicate the order of the surface in each of the u and v parameter dimensions. **PEXGetImpDepConstants** can be called to determine the largest supported value for surface u and v order. The spline shape is specified using two lists of knots in the parametric coordinate space, plus an array of control points specified in modeling coordinates. The u and v knot sequences must each form a non-decreasing sequence of numbers. The column count indicates the number of control points in the u direction and the row count indicates the number of control points in the v direction. The control points are stored in the array in row-major order (i.e., the column number varies fastest as vertices are stored in the array) and the rows increase in the direction of increasing v. The number of knots in the u direction is the sum of the order in the u direction and column count. The number of knots in the v direction is the sum of the order in the v direction and row count. The number of control points in each direction must be at least as large as the corresponding order.

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

If the rationality is **PEXRational**, the control point list must be specified in homogeneous (4D) modeling coordinates. If rationality is **PEXNonRational**, the control point list must be specified in non-homogeneous (3D) modeling coordinates.

In addition to the parametric bounds, a list of trimming loops may also be specified. Trimming loops serve to further restrict the region in parametric coordinate space over which the surface is evaluated. Each trimming loop is defined as a list of one or more B-spline trimming curves that are connected head-to-tail. Each trim curve is a completely specified NURB curve, i.e. it is rational or non-rational, has its own order, etc. The list must be explicitly closed so that the tail of the last curve joins the head of the first. Each trimming curve is parameterized independently. If there is floating point inaccuracy in closure or in head-to-tail connectivity between curves, closure or connectivity will be assumed. Trimming loops are defined in the parameter space of the surface and may not go outside the parameter space of the surface.

When no trimming loops are specified, the rectangular parameter limits of the surface are renderer as the edges of the surface based on the edge flag attribute.

Trimming loops define the region of the surface that is to be rendered based on the following two rules: (1) a point is in the portion of the surface to be rendered if any ray projected from it to infinity has an odd number of intersections with trimming loops, and (2) traveling in the direction of a trimming loop, the portion of the surface to be trimmed away should be on the right and the portion to be retained should be on the left. In other words, a loop defined in counter-clockwise order will cause the interior of the loop to be retained and the exterior to be trimmed away. A clockwise loop will cause the exterior of the loop to be retained and the interior to be trimmed away. If loops are nested, they must alternate in direction. In all cases, the outermost loop must be counter-clockwise. No trimming curve may intersect itself and no trimming loop may intersect itself or any other trimming loop. Trimming loops that do not obey these rules will result in implementation-dependent behavior.

Each trimming curve has a visibility flag that controls its visibility for the purposes of surface edge display. Depending on the surface edge attributes and the visibility flags associated with trimming curves, the curves in trimming loops may be drawn as surface edges.

All attributes affecting the representation of fill area sets also affect the representation of the nonunform B-spline surface primitive. In addition, the surface approximation is used to determine how to approximate the B-spline surface and the parametric surface characteristics are used to specify the appearance of the surface.

If either of the specified surface orders are not supported, the output primitive is stored in a structure, but when rendered, the primitive is ignored and has no visual effect.

Trimming curve specification must abide by the constraints of NURB curve (e.g. number of control points at least as large as the order, non-decreasing knot sequence, order plus number of controls points equals the number of knots). Also, the trim curve order must not be less than two.

Data Structures

```
typedef union {
    PEXCoord2D          *point_2d;
    PEXCoord            *point;
    PEXCoord4D          *point_4d;
} PEXArrayOfCoord;      /* Pointer to array of points */

typedef struct {
    unsigned short      count;                  /* number of curves */
    PEXTrimCurve        *curves;
} PEXListOfTrimCurve;

typedef struct {
    PEXSwitch           visibility;       /* True or False */
    unsigned char       reserved;
    unsigned short      order;
    PEXCoordType        rationality;      /* PEXRational or PEXNonRational */
    PEXEnumTypeIndex    approx_method;        /* see PEXGetEnumTypeInfo */
```

```
    float               tolerance;
    float               tmin, tmax;
    PEXListOfFloat      knots;
    unsigned short      count;                  /* number of control points */
    PEXArrayOfCoord     control_points;
} PEXTrimCurve;

typedef unsigned char   PEXSwitch;
typedef unsigned short  PEXCoordType;
typedef short           PEXEnumTypeIndex;

typedef struct {
    unsigned short      count;                  /* number of floats */
    float               *floats;
} PEXListOfFloat;
```

See also the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**, **PEXSetSurfaceApprox**, **PEXSetFacetCullingMode**, **PEXSetFacetDistinguishFlag**, **PEXSetPatternSize**, **PEXSetPatternAttributes**, **PEXSetPatternAttributes2D**, **PEXSetInteriorBundleIndex**, **PEXSetSurfaceEdgeFlag**, **PEXSetSurfaceEdgeType**, **PEXSetSurfaceEdgeWidth**, **PEXSetSurfaceEdgeColor**, **PEXSetSurfaceEdgeColorIndex**, **PEXSetEdgeBundleIndex**, **PEXSetParaSurfCharacteristics**, **PEXGetImpDepConstants**

PEXPolyline - 3D Polyline Primitive

Synopsis

void PEXPolyline(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *count*, PEXCoord **points*)

Arguments

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	The number of points.
points	An array of points defining the polyline.

Returns

None

Description

This function creates a polyline output primitive.

The points, specified in modeling coordinates, are joined together by line segments. The first point is joined to the second, the second to the third, and so on. The last point is not joined to the first.

The polyline colors are affected only by depth-cueing and are mapped to device colors. Polylines are not displayed if they are outside the currently defined clipping volume. Polylines crossing the clipping volume are clipped and only the portions inside the clipping volume are displayed.

Depending on the ASF attributes, the line color, line type, line width, and polyline interpolation method attributes are obtained either directly from the current attributes or from the line bundle.

A polyline with fewer than two points is considered degenerate. It is stored in a structure, but when rendered, the primitive is ignored and has no visual effect.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetLineType, **PEXSetLineWidth**,
PEXSetLineColorIndex, **PEXSetLineColor**, **PEXSetLineBundleIndex**

PEXPolyline2D - 2D Polyline Primitive

Synopsis

void PEXPolyline2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *count*, PEXCoord2D **points*)

Arguments

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	The number of points.
points	An array of points defining the polyline.

Returns

None

Description

This function creates a 2D polyline output primitive.

This function is like **PEXPolyline**, except that the points consist of only x- and y-components. The z-component is assumed to be zero. This primitive is two-dimensional only in that the z-components are implied. Geometry transformations are still carried out in three dimensions.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetLineType, **PEXSetLineWidth**,
PEXSetLineColorIndex, **PEXSetLineColor**, **PEXSetLineBundleIndex**

PEXPolylineSetWithData - 3D Set of Polylines Primitive

Synopsis

void PEXPolylineSetWithData(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *vertex_attributes*, int *color_type*, unsigned int *count*, PEXListOfVertex **vertex_lists*)

Arguments

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_attributes A mask indicating the vertex attributes provided (**PEXGANone** or **PEXGAColor**).

color_type The type of color data provided (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).

count The number of polylines.

vertex_lists A pointer to a list of vertex arrays defining each polyline in the set.

Returns

None

Description

This function creates a polyline set output primitive.

This function is similar to **PEXPolyline** except that it allows applications to specify a number of polylines as well as additional data. These polylines may contain a color value at each vertex. The vertex attributes indicate the content of the additional data. Color values passed must be of the specified color type.

If colors are passed for each vertex, they are used instead of the line color attribute. The polyline interpolation method, which depending on the ASF attribute is obtained directly from the current polyline interpolation attribute or from the line bundle, controls how the polylines are shaded.

All other aspects of this primitive are the same as **PEXPolyline**.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetLineType, **PEXSetLineWidth**, **PEXSetLineColorIndex**, **PEXSetLineColor**, **PEXSetPolylineInterpMethod**, **PEXSetLineBundleIndex**

PEXQuadrilateralMesh - 3D Quadrilateral Mesh Primitive

Synopsis

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

Arguments

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 quadrilaterals (**PEXShapeComplex**, **PEXShapeNonConvex**, **PEXShapeConvex**, **PEXShapeUnknown**).
facet_attributes	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
vertex_attributes	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
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

This function creates a quadrilateral mesh output primitive.

The quadrilateral mesh surface is created from the vertices. The vertex array is accessed in row major order (i.e. the column number varies fastest as vertices are accessed). 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.

Normals for quadrilaterals, if not provided explicitly, are computed by taking the cross product of the diagonals. For a quadrilateral with the above vertices, the cross product would be formed as follows:

normal(i,j) = (point(i+1, j+1) - point(i, j)) X (point(i, j+1) - point(i+1, j))

Normals are assumed to be unit length vectors. If not unit length, the result is implementation-dependent.

There must be an entry in the facet data array for each facet, if facet data is indicated by the facet attributes.

All other aspects of this primitive are the same as **PEXFillAreaSetWithData**.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXSetOfFillAreaSets - 3D Set of Fill Area Sets Primitive

Synopsis

void PEXSetOfFillAreaSets(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *shape_hint*, 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*, PEXArrayOfVertex *vertices*, unsigned int *index_count*, PEXSwitch **edge_flags*, PEXConnectivityData **connectivity*)

Arguments

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**).
facet_attributes	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
vertex_attributes	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
edge_attributes	A mask indicating the edge attributes provided (**PEXGANone** or **PEXGAEdges**).
contour_hint	A flag indicating 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

This function creates a set of fill area sets output primitive.

A fill area may cross over itself to create a complex shape. The odd-even rule is used for determining the area that lies in the interior of the fill area. The shape hint is provided to enable performance improvements for certain shapes. Fill areas that are of higher complexity than indicated by the shape hint are rendered in an implementation-dependent manner. Consequently, applications should pass **PEXShapeUnknown** as the shape unless they are certain the fill area's shape is one of the other three. Note that a fill area set with more than one contour is always allowed to have contours that intersect. If is quite possible that the only times rendering optimization can occur are when the number of contours in a fill area set is equal to one or if the "contours all one" flag is **True**, and the shape hint is **PEXShapeConvex**.

The contour hint provides further information about the relationships between contours in the fill area set. If the contour hint is **PEXContourDisjoint**, all contours will be spatially disjoint. No overlapping or intersection occurs between any contours in the fill area set. If the contour hint is **PEXContourNested**, 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 the contour hint is **PEXContourIntersecting**, separated contours may have edges that are coincident or overlap. If the contour hint is **PEXContourUnknown** nothing is known about the interrelationships between contours. Fill area sets with contours that have higher complexity interrelationships than that indicated by the contour hint are rendered in an implementation-dependent manner.

The facet attributes indicate the content of the facet data. This data may be a color, a normal, or a color followed by a normal. Use the constants **PEXGANone**, **PEXGAColor** and **PEXGANormal** to construct a mask indicating the data provided. If specified, the facet color takes precedence over the surface color. If specified, the facet normal is used to determine whether the fill area is backfacing.

The vertex attributes indicate the content of each fill area vertex. In addition to the coordinate (x,y,z), applications may specify a color, a normal, or a color followed by a normal for each vertex. Use the constants **PEXGANone**, **PEXGAColor** and **PEXGANormal** to construct a mask indicating the data provided. If specified, vertex colors will override facet color or the current surface color. If specified, vertex normals are taken to be normals at the vertices of the fill area.

The reflection model and the surface interpolation will affect how the additional data is used in rendering the surface.

Color values passed must be of the specified color type. Normals are assumed to be unit length vectors. The effect if the normal is not unit length is implementation-dependent.

The edge attributes indicate the content of the edge flags. Use **PEXGAEdges** to indicate edge flags are provided or **PEXGANone** if no edge flags are provided. The edge flags, if present, are set to **PEXOn** or **PEXOff** and are used to indicate which edges should be rendered. The edge control for vertex i indicates whether or not to render the edge between vertex i and vertex i+1. Surface edges are always rendered with the surface edge color and are not affected by the facet or vertex colors.

The connectivity of the primitive is defined by the connectivity list. The connectivity list is a pointer to an array of **PEXConnectivityData** structures. Each entry in the array gives the contours for one fill area set in the set of fill area sets, and, in turn, contains a pointer to an array of **PEXListOfUShort** structures. Each of these latter structures gives the index of the vertices of one contour in that fill area set. The indices select a vertex in the array of vertices. Vertices are numbered with indices starting from zero (i.e. the first vertex is referenced as vertex 0). As a special case, if the "contours all one" flag is **True** then the contour count field in each fill area set is guaranteed to be one.

All attributes affecting the representation of fill area sets also affect the representation of this primitive.

Data Structures

```
typedef struct {
    unsigned short      count;                  /* number of lists */
    PEXListOfUShort     *lists;
} PEXConnectivityData;

typedef struct {
    unsigned short      count;                  /* number of shorts */
    unsigned short      *shorts;
```

```
} PEXListOfUShort;

typedef unsigned char   PEXSwitch;
```

See also the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

PEXText - 3D Text Primitive

Synopsis

void PEXText(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **origin*, PEXVector **vector1*, PEXVector **vector2*, int *length*, char **string*)

Arguments

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**).
origin	The origin of the text string.
vector1	A vector defining the positive x-direction of the text local coordinate system.
vector2	A vector defining the positive y-direction of the text local coordinate system.
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None

Description

This function creates a text output primitive.

The first character set in the text font will be used.

The text string is located on a plane defined by its position and direction vectors. The origin defines the position, in model coordinates, at which to render the text string. The two direction vectors define the positive x- and y-directions of the text local coordinate system. If the two vectors are parallel or if one of the vectors has zero length, the vector values <1,0,0> and <0,1,0> are used.

During rendering, the string's position is transformed to a position in device coordinates. The string's color is only affected by depth-cueing and is mapped to a device color. The text string is clipped depending on the current text precision attribute. If the text precision is **PEXStringPrecision**, clipping is done in an implementation-dependent fashion. If the text precision is **PEXCharPrecision**, clipping is done on at least a character-by-character basis. If the text precision is **PEXStrokePrecision**, clipping is performed at the clipping boundaries for each character.

Depending on the text ASF values, the text color, text precision, character expansion, character spacing, and text font attributes are obtained either directly from the current text attribute values or from the entry in the text bundle specified by the current text bundle index attribute. The current character height, text path, text alignment attributes and character up vector are also used to render the text string. The directions specified by the character up vector and text path are relative to the text local coordinate system.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetCharHeight**, **PEXSetCharUpVector**, **PEXSetTextPath**, **PEXSetTextAlignment**, **PEXSetTextBundleIndex**

PEXText2D - 2D Text Primitive

Synopsis

void PEXText2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord2D **origin*, int *length*, char **string*)

Arguments

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**).
origin	The origin of the text string.
length	The number of bytes in the text string.
string	A pointer to the text string.

Returns

None

Description

This function creates a 2D text output primitive.

This function is like **PEXText**, except that the origin position consists of only x- and y-components. The z-component is assumed to be zero. This primitive is two-dimensional only in that the z-component is implied. Geometry transformations are still carried out in three dimensions.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextFontIndex, **PEXSetTextPrecision**, **PEXSetCharExpansion**, **PEXSetCharSpacing**, **PEXSetTextColorIndex**, **PEXSetTextColor**, **PEXSetCharHeight**, **PEXSetCharUpVector**, **PEXSetTextPath**, **PEXSetTextAlignment**, **PEXSetTextBundleIndex**

PEXTriangleStrip - 3D Triangle Strip Primitive

Synopsis

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

Arguments

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**).
facet_attributes	A mask indicating the facet attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
vertex_attributes	A mask indicating the vertex attributes provided (**PEXGANone**, **PEXGAColor**, **PEXGANormal**).
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

This function creates a triangle strip output primitive.

The triangle strip is created from the array of vertices. There are two less facets in the strip than the number of vertices. The first triangle in the strip 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. There must be an entry in the facet data array for each facet, if facet data is indicated by the facet attributes.

Normals are assumed to be unit length vectors. If not unit length, the result is implementation-dependent.

A triangle strip with fewer than three vertices is considered degenerate. It is stored in a structure, but when rendered, the primitive is ignored and has no visual effect.

All other aspects of this primitive are the same as **PEXFillAreaWithData**.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

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**

2.3. Output Primitive Attribute Functions

This section describes all the output primitive attribute functions. More detailed information on attributes is given in *Chapter 4: Pipeline Contexts*. Attributes that have optional enumerated types have their valid values listed under **PEXGetEnumTypeInfo** in *Chapter 1: Introduction to PEXlib*.

2.3.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* coordinates */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXCoord;

      typedef struct {
          float               x;
          float               y;
      } PEXCoord2D;

/* vectors */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXVector;

      typedef struct {
          float               x;
          float               y;
      } PEXVector2D;

/* colors */

      typedef union {
          PEXColorIndexed     indexed;
          PEXColorRGB         rgb;
          PEXColorHSV         hsv;
          PEXColorHLS         hls;
          PEXColorCIE         cie;
          PEXColorRGB8        rgb8;
          PEXColorRGB16       rgb16;
      } PEXColor;

      typedef struct {
          PEXTableIndex       index;
          unsigned short      reserved;
      } PEXColorIndexed;

      typedef struct {
          float               red;
          float               green;
          float               blue;
      } PEXColorRGB;
```

```
typedef struct {
    float               hue;
    float               saturation;
    float               value;
} PEXColorHSV;

typedef struct {
    float               hue;
    float               lightness;
    float               saturation;
} PEXColorHLS;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXColorCIE;

typedef struct {
    unsigned char       red;
    unsigned char       green;
    unsigned char       blue;
    unsigned char       reserved;
} PEXColorRGB8;

typedef struct {
    unsigned short      red;
    unsigned short      green;
    unsigned short      blue;
    unsigned short      reserved;
} PEXColorRGB16;

typedef struct {
    PEXColorType        type;/* PEXColorTypeIndexed, PEXColorTypeRGB, */
                         /* PEXColorTypeCIE, PEXColorTypeHSV, */
                         /* PEXColorTypeHLS, PEXColorTypeRGB8, */
                         /* PEXColorTypeRGB16 */
    unsigned short      reserved;
    PEXColor            value;
} PEXColorSpecifier;
```

```
/* matrices */

      typedef float           PEXMatrix[4][4];
      typedef float           PEXMatrix3x3[3][3];

/* similar to XErrorEvent - use to access additional info in OC
error */

      typedef struct {
          int             type;
          Display         *display;      /* Display the event was read from */
          XID             resourceid;    /* resource id of renderer or structure */
          unsigned long   serial;        /* serial number of failed request */
          unsigned char   error_code;    /* error code of failed request */
          unsigned char   request_code;  /* Major op-code of failed request */
```

```
    unsigned char   minor_code;     /* Minor op-code of failed request */
    unsigned short  op_code;        /* op-code of failed output command */
    unsigned short  count;          /* number of output commands successfully */
                                    /* executed before error */
} PEXOCErrorEvent;
```

PEXAddToNameSet - Add Names to Name Set

Synopsis

void PEXAddToNameSet(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned long *count*, PEXName **names*)

Arguments

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	The number of names.
names	An array of names to be added to the current name set.

Returns

None

Description

This function creates an output primitive attribute which adds the specified list of names to the current name set. If any name is outside the supported range that name is ignored.

Data Structures

```
typedef unsigned long   PEXName;
```

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateNameSet, **PEXRemoveFromNameSet**, **PEXGetImpDepConstants**

PEXRemoveFromNameSet - Remove Names from Name Set

Synopsis

void PEXRemoveFromNameSet(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned long *count*, PEXName **names*)

Arguments

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	The number of names.
names	An array of names to be removed from the current name set.

Returns

None

Description

This function creates an output primitive attribute which removes the specified list of names from the current name set. If any name is outside the supported range that name is ignored.

Data Structures

```
typedef unsigned long   PEXName;
```

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateNameSet, **PEXAddToNameSet**, **PEXGetImpDepConstants**

PEXRestoreModelClipVolume - Restore Model Clip Volume

Synopsis

void PEXRestoreModelClipVolume(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*)

Arguments

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**).

Returns

None

Description

This function creates an output primitive attribute which restores the last-saved modeling clipping volume. If there is no last-saved clipping volume that can be restored, the modeling clipping volume is restored to its default state.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetModelClipFlag, **PEXSetModelClipVolume**, **PEXSetModelClipVolume**

PEXSetATextAlignment - Set Annotation Text Alignment

Synopsis

void PEXSetATextAlignment(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*,
int *halignment*, int *valignment*)

Arguments

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**).

halignment — The horizontal annotation text alignment (**PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignCenter**, **PEXHAlignRight**).

valignment — The vertical annotation text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase**, **PEXVAlignBottom**).

Returns

None

Description

This function creates an output primitive attribute which sets the horizontal and vertical annotation text alignment attributes.

Errors

BadPEXOutputCommand — The output command contains an invalid value.

BadPEXRenderer — The specified renderer resource identifier is invalid.

BadPEXStructure — The specified structure resource identifier is invalid.

See Also

PEXSetATextHeight - Set Annotation Text Height

Synopsis

void PEXSetATextHeight(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *height*)

Arguments

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**).
height	The annotation text character height.

Returns

None

Description

This function creates an output primitive attribute which sets the annotation text height attribute. If the specified height or the computed width is not supported, the height or width is mapped to the nearest supported annotation character height or width. These values depend on the font files that are in the font groups in the selected font table entry, which in turn depend on which X or PEX font files have been opened. If all scalable and rotatable stroke fonts are open, then a continuous range of character sizes are supported. The height is expressed in normalized projection coordinates.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetATextPath - Set Annotation Text Path

Synopsis

void PEXSetATextPath(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *path*)

Arguments

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**).
path	The text (drawing) path (**PEXPathRight**, **PEXPathLeft**, **PEXPathUp**, **PEXPathDown**).

Returns

None

Description

This function creates an output primitive attribute which sets the annotation text path attribute.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetATextStyle - Set Annotation Text Style

Synopsis

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

Arguments

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 annotation text style (**PEXATextNotConnected** or **PEXATextConnected**).

Returns

None

Description

This function creates an output primitive attribute which sets the annotation text style attribute. If the specified style is not supported, **PEXATextNotConnected** is used. Supported values for annotation text style are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo

PEXSetATextUpVector - Set Annotation Text Up Vector

Synopsis

void PEXSetATextUpVector(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXVector2D **vector*)

Arguments

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**).
vector	The annotation text character up vector.

Returns

None

Description

This function creates an output primitive attribute which sets the annotation text up vector attribute. The vector is specified in the text local coordinate system defined by the direction vectors associated with the annotation text primitive. Only the direction of the vector is used, not the magnitude. However, if the annotation text up vector is degenerate (it has a length of zero), a value <0,1> is used.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetBFInteriorStyle - Set Back-Face Surface Interior Style

Synopsis

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

Arguments

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 back-facing surface interior style (**PEXInteriorStyleHollow**, **PEXInteriorStyleSolid**, **PEXInteriorStylePattern**, **PEXInteriorStyleHatch**, **PEXInteriorStyleEmpty**).

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing surface interior style attribute. If the specified style is not supported, **PEXInteriorStyleHollow** is used. Supported values for back-facing surface interior style are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetBFInteriorStyleIndex - Set Back-Face Surface Interior Style Index

Synopsis

void PEXSetBFInteriorStyleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *index*)

Arguments

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**).
index	The back-facing surface interior style index.

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing interior style index attribute. If the current back-facing interior style is **PEXInteriorStylePattern** or **PEXInteriorStyleHatch**, the specified index is used to further define the rendering style of back-facing surface primitives. For **PEXInteriorStylePattern**, if the specified pattern table index is not defined, table index one is used. For **PEXInteriorStyleHatch**, the index determines the hatch style. If the specified hatch style is not supported, style one is used. If style one is not supported, the result is implementation-dependent. Supported values for hatch style are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**, **PEXCreateLookupTable**

PEXSetBFReflectionAttributes - Set Back-Face Surface Reflection Attributes

Synopsis

void PEXSetBFReflectionAttributes(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXReflectionAttributes **attributes*)

Arguments

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**).
attributes	The back-facing surface reflection attributes.

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing surface reflection attributes. Surface reflection attributes consist of the ambient, diffuse and specular coefficients, the specular concentration and color, and the transmission coefficient.

Data Structures

```
typedef struct {
    float               ambient;
    float               diffuse;
    float               specular;
    float               specular_conc;
    float               transmission;
    PEXColorSpecifier   specular_color;
} PEXReflectionAttributes;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXGetImpDepConstants**

PEXSetBFReflectionModel - Set Back-Face Surface Reflection Model

Synopsis

void PEXSetBFReflectionModel(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *model*)

Arguments

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**).
model	The back-facing surface reflection model (**PEXReflectionNone**, **PEXReflectionAmbient**, **PEXReflectionDiffuse**, **PEXReflectionSpecular**).

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing surface reflection model attribute. If the specified reflection model is not supported, **PEXReflectionNone** is used. Supported values for back-facing surface reflection model are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetBFSurfaceColor - Set Back-Face Surface Color

Synopsis

void PEXSetBFSurfaceColor(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *color_type*, PEXColor **color*)

Arguments

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**).
color_type	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
color	A pointer to the back-facing surface color.

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing surface color attribute. The attribute is set to either an indexed color or a direct color value, depending on the color type. If the color type is **PEXColorTypeIndexed** and the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value, e.g. the color type is **PEXColorTypeIndexed**, and the color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetBFSurfaceColorIndex - Set Back-Face Surface Color Index

Synopsis

void PEXSetBFSurfaceColorIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The color table index for back-facing surfaces.

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing surface color attribute to an indexed color with the value indicated by index. If the color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand
The color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetBFSurfaceInterpMethod - Set Back-Face Surface Interpolation Method

Synopsis

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

Arguments

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	The back-facing surface interpolation method (**PEXSurfaceInterpNone**, **PEXSurfaceInterpColor**, **PEXSurfaceInterpDotProduct**, **PEXSurfaceInterpNormal**).

Returns

None

Description

This function creates an output primitive attribute which sets the back-facing surface interpolation method attribute. If the specified interpolation method is not supported, **PEXSurfaceInterpNone** is used. Supported values for back-facing surface interpolation method are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetCharExpansion - Set Character Expansion Factor

Synopsis

void PEXSetCharExpansion(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *expansion*)

Arguments

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**).
expansion	The character expansion factor.

Returns

None

Description

This function creates an output primitive attribute which sets the character expansion factor attribute. Only the magnitude of the specified expansion is considered. The specified character expansion factor is compared to the minimum and maximum character expansion factors. These values depend on the font files that are in the font groups in the selected font table entry, which in turn depend on the font files opened. For example, if all scalable and rotatable stroke fonts are open, then a continuous number of expansions are supported. If the expansion is smaller than the minimum character expansion factor, the minimum value is used. If the expansion is larger than the maximum character expansion factor, the maximum value is used.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF

PEXSetCharHeight - Set Character Height

Synopsis

void PEXSetCharHeight(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *height*)

Arguments

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**).

height The text character height.

Returns

None

Description

This function creates an output primitive attribute which sets the character height value attribute. If the specified height or the computed width is not supported, the height or width is mapped to the nearest supported character height or width. These values depend on the font files that are in the font groups in the selected font table entry, which in turn depend on which X or PEX fonts are open. For example, if the client opens all scalable and rotatable stroke fonts, a continuous number of character sizes are supported. The height is specified in the text local coordinate system.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetCharSpacing - Set Character Spacing

Synopsis

void PEXSetCharSpacing(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *spacing*)

Arguments

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**).
spacing	The text character spacing factor.

Returns

None

Description

This function creates an output primitive attribute which sets the character spacing attribute. The character spacing attribute is expressed as a fraction of the font's nominal character height.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF

PEXSetCharUpVector - Set Character Up Vector

Synopsis

void PEXSetCharUpVector(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXVector2D **vector*)

Arguments

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**).
vector	The text character up vector.

Returns

None

Description

This function creates an output primitive attribute which sets the text up vector attribute. The vector is specified in the text local coordinate system defined by the direction vectors associated with the annotation text primitive. If the annotation text up vector is degenerate (it has a length of zero), a value <0,1> is used.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetColorApproxIndex - Set Color Approximation Index

Synopsis

void PEXSetColorApproxIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*,
unsigned int *index*)

Arguments

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**).
index	The color approximation table index.

Returns

None

Description

This function creates an output primitive attribute which sets the color approximation index attribute. If the specified index is not defined, index zero is used. If index zero is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand	The color approximation index value exceeds 65534.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXCreateLookupTable, **PEXGetImpDepConstants**

PEXSetCurveApprox - Set Curve Approximation Method

Synopsis

void PEXSetCurveApprox(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *method*, double *tolerance*)

Arguments

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	The curve approximation method (**PEXApproxImpDep**, **PEXApproxConstantBetweenKnots**, **PEXApproxWCChordalSize**, **PEXApproxNPCChordalSize**, **PEXApproxDCChordalSize**, **PEXCurveApproxWCChordalDev**, **PEXCurveApproxNPCChordalDev**, **PEXCurveApproxDCChordalDev**, **PEXApproxWCRelative**, **PEXApproxNPCRelative**, **PEXApproxDCRelative**).
tolerance	The curve approximation tolerance (specific to each method).

Returns

None

Description

This function creates an output primitive attribute which sets the curve approximation attribute. If the specified method is not supported, an implementation-dependent method (method 1) is used. Supported values for curve approximation are inquirable via **PEXGetEnumTypeInfo**. The tolerance value is provided to indicate the desired accuracy of the approximation, and is used in different ways for the different methods. See *Chapter 4: Pipeline Contexts* for more information about tolerance values.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXGetEnumTypeInfo**

PEXSetDepthCueIndex - Set Depth Cue Index

Synopsis

void PEXSetDepthCueIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The depth cue table index.

Returns

None

Description

This function creates an output primitive attribute which sets the depth-cue index attribute. If the specified index is not defined, index zero is used. If index zero is not defined, depth cueing is turned off.

Errors

BadPEXOutputCommand
The depth cue index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateLookupTable

PEXSetEdgeBundleIndex - Set Edge Bundle Index

Synopsis

void PEXSetEdgeBundleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The edge bundle table index.

Returns

None

Description

This function creates an output primitive attribute which sets the edge bundle index attribute. If an undefined index is specified, default bundle index one is used. If index one is not defined, the edge bundle attributes are set as follows: surface edges are turned off, the edge type is solid, the edge width is 1.0 and the color is indexed color one.

An edge bundle table index of 0 will produce a **BadPEXOutputCommand** error.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetFacetCullingMode - Set Facet Culling Mode

Synopsis

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

Arguments

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 facet culling mode (**PEXNone**, **PEXBackFaces**, **PEXFrontFaces**).

Returns

None

Description

This function creates an output primitive attribute which sets the facet culling mode attribute.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetFacetDistinguishFlag - Set Facet Distinguish Flag

Synopsis

void PEXSetFacetDistinguishFlag(Display *display, XID resource_id, PEXOCRequestType req_type, int flag)

Arguments

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**).
flag	The facet distinguish flag (**True** or **False**).

Returns

None

Description

This function creates an output primitive attribute which sets the facet distinguish flag attribute. Values for the facet distinguish flag are **True** (use back-face attributes to renderer surfaces) or **False** (use front-face attributes to renderer surfaces).

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetGlobalTransform - Set Global Transformation 3D

Synopsis

void PEXSetGlobalTransform(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*,
PEXMatrix *transform*)

Arguments

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**).
transform	The new global transformation matrix.

Returns

None

Description

This function creates an output primitive attribute which replaces the current global transformation matrix with the specified matrix. The matrix is stored in row-major order.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXSetGlobalTransform2D

PEXSetGlobalTransform2D - Set Global Transformation 2D

Synopsis

void PEXSetGlobalTransform2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXMatrix3x3 *transform*)

Arguments

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**).
transform	The new global transformation matrix.

Returns

None

Description

This function creates an output primitive attribute which replaces the current global transformation matrix with the specified matrix. This output command is similar to **PEXSetGlobalTransform** except that the global transformation matrix is specified as a 3×3 matrix. Before replacement of the global transformation matrix, the 3×3 matrix represented by

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

is expanded to a 4×4 matrix as follows:

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

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetGlobalTransform

PEXSetHLHSRID - Set Hidden-Line Hidden-Surface Removal Identifier

Synopsis

void PEXSetHLHSRID(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned long *hlhsr_id*)

Arguments

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**).
hlhsr_id	The HLHSR identifier (**PEXHLHSRIDDisable** or **PEXHLHSRIDEnable**).

Returns

None

Description

This function creates an output primitive attribute which sets the HLHSR identifier attribute. This output command is a no-op for all standard HLHSR modes except **PEXHLHSRZBufferID**. If the renderer's HLHSR mode is set to **PEXHLHSRZBufferID**, then a HLHSR identifier of **PEXHLHSRIDDisable** will disable z-buffering and a HLHSR identifier of **PEXHLHSRIDEnable** will enable z-buffering. For non-standard HLHSR modes, the effect of this output command is implementation-dependent.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF - Set Individual Aspect Source Flags

Synopsis

void PEXSetIndividualASF(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned long *attribute*, int *asf*)

Arguments

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**).
attribute	The ASF attribute name (see the Description).
asf	The attribute source flag value (**PEXIndividual** or **PEXBundled**).

Returns

None

Description

This function creates an output primitive attribute which sets the aspect source flag for a single, individual attribute. For the individual ASF attribute name, valid values are:

PEXASFBFInteriorStyle
PEXASFBFInteriorStyleIndex
PEXASFBFReflectionAttr
PEXASFBFReflectionModel
PEXASFBFSurfaceColor
PEXASFBFSurfaceInterp
PEXASFCharExpansion
PEXASFCharSpacing
PEXASFCurveApprox
PEXASFInteriorStyle
PEXASFInteriorStyleIndex
PEXASFLineColor
PEXASFLineType
PEXASFLineWidth
PEXASFMarkerColor
PEXASFMarkerScale
PEXASFMarkerType
PEXASFPolylineInterp
PEXASFReflectionAttr
PEXASFReflectionModel
PEXASFSurfaceApprox
PEXASFSurfaceColor
PEXASFSurfaceEdgeColor
PEXASFSurfaceEdgeType
PEXASFSurfaceEdgeWidth
PEXASFSurfaceEdges
PEXASFSurfaceInterp
PEXASFTextColor
PEXASFTextFontIndex
PEXASFTextPrec

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetInteriorBundleIndex - Set Interior Bundle Index

Synopsis

void PEXSetInteriorBundleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).

index The interior bundle table index.

Returns

None

Description

This function creates an output primitive attribute which sets the interior bundle index attribute. If the specified index is not defined, the default index one is used. If index one is not defined, the following bundle values are used: interior style is **PEXInteriorStyleHollow**; interior style index is one; surface color is indexed color one; reflection attributes are ambient, diffuse and specular coefficient of 1.0, specular concentration and transmission coefficient of 0.0 and specular color of indexed color one; reflection model is **PEXReflectionNone**; surface interpolation is **PEXSurfaceInterpNone**; and surface approximation is method 1 (**PEXApproxImpDep**) with u and v tolerances of 1.0. The back-facing attributes are set to identical values as the front-facing attributes.

An interior bundle table index of 0 will produce a **BadPEXOutputCommand** error.

Errors

BadPEXOutputCommand The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetInteriorStyle - Set Surface Interior Style

Synopsis

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

Arguments

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 interior style (**PEXInteriorStyleHollow**, **PEXInteriorStyleSolid**, **PEXInteriorStylePattern**, **PEXInteriorStyleHatch**, **PEXInteriorStyleEmpty**).

Returns

None

Description

This function creates an output primitive attribute which sets the surface interior style attribute. If the specified style is not supported, **PEXInteriorStyleHollow** is used. Supported values for surface interior style are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetInteriorStyleIndex - Set Surface Interior Style Index

Synopsis

void PEXSetInteriorStyleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *index*)

Arguments

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**).
index	The interior style index.

Returns

None

Description

This function creates an output primitive attribute which sets the interior style index attribute. If the current interior style is **PEXInteriorStylePattern** or **PEXInteriorStyleHatch**, the specified index is used to further define the rendering style of front-facing surface primitives. For **PEXInteriorStylePattern**, if the specified pattern table index is not defined, table index one is used. For **PEXInteriorStyleHatch**, the index determines the hatch style. If the specified hatch style is not supported, style one is used. If style one is not supported, the result is implementation-dependent. Supported values for hatch style are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**, **PEXGetEnumTypeInfo**

PEXSetLightSourceState - Set Light Source State

Synopsis

void PEXSetLightSourceState(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *enable_count*, PEXTableIndex **enable*, unsigned int *disable_count*, PEXTableIndex **disable*)

Arguments

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**).
enable_count	The number of lights to enable.
enable	An array of lights to enable.
disable_count	The number of lights to disable.
disable	An array of lights to disable.

Returns

None

Description

This function creates an output primitive attribute which sets the light source state attribute. Each element in the enable list activates the light represented by the corresponding light table entry and each element in the disable list deactivates the light represented by the corresponding light table entry.

A light listed in both the enable list and the disable list, or a light index of 0 will produce a **BadPEXOutputCommand** error. Otherwise, if any light in the enable or disable list references an undefined light table entry, the light is ignored.

Data Structures

```
typedef unsigned short  PEXTableIndex;
```

Errors

BadPEXOutputCommand	The output command contains an invalid value.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXCreateLookupTable

PEXSetLineBundleIndex - Set Line Bundle Index

Synopsis

void PEXSetLineBundleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The line bundle table index.

Returns

None

Description

This function creates an output primitive attribute which sets the line bundle index attribute. If an undefined line bundle index is specified, the default bundle index one is used. If index one is not defined, the following values are used: line type is **PEXLineTypeSolid**; polyline interpolation is **PEXPolylineInterpNone**; curve approximation is method 1 (**PEXApproxImpDep**) with a tolerance of 1.0; line width is 1.0 and line color is index color one.

A line bundle table index of 0 will produce a **BadPEXOutputCommand** error.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetLineColor - Set Line Color

Synopsis

void PEXSetLineColor(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *color_type*, PEXColor **color*)

Arguments

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**).
color_type	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
color	A pointer to the line color.

Returns

None

Description

This function creates an output primitive attribute which sets the line color attribute. The attribute is set to either an indexed color or a direct color value, depending on the color type. If the color type is **PEXColorTypeIndexed** and the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value, e.g. the color type is **PEXColorTypeIndexed**, and the color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetLineColorIndex - Set Line Color Index

Synopsis

void PEXSetLineColorIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The color table index for lines.

Returns

None

Description

This function creates an output primitive attribute which sets the line color attribute to an indexed color with the value indicated by index. If the color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand	The color index value exceeds 65534.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetLineType - Set Line Type

Synopsis

void PEXSetLineType(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *line_type*)

Arguments

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**).
line_type	The line type (**PEXLineTypeSolid**, **PEXLineTypeDashed**, **PEXLineTypeDotted**, **PEXLineTypeDashDot**).

Returns

None

Description

This function creates an output primitive attribute which sets the line type attribute. If the specified line type is not supported, **PEXLineTypeSolid** is used. Supported values for line type are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetLineWidth - Set Line Width

Synopsis

void PEXSetLineWidth(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *width*)

Arguments

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**).
width	The line width scale factor.

Returns

None

Description

This function creates an output primitive attribute which sets the line width attribute. The specified width is used as a scale factor. This scale factor is used to increase or decrease the width (in pixels) from the nominal line width for the display device. The result is mapped to the nearest supported line width.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, **PEXSetIndividualASF**

PEXSetLocalTransform - Set Local Transformation 3D

Synopsis

void PEXSetLocalTransform(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *composition*, PEXMatrix *transform*)

Arguments

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**).
composition	The composition rule for combining the new matrix with the current local transform (**PEXPreConcatenate**, **PEXPostConcatenate**, **PEXReplace**).
transform	The new local transformation matrix.

Returns

None

Description

This function creates an output primitive attribute which modifies the local transformation matrix. If the composition type is **PEXPreConcatenate**, the specified matrix is pre-concatenated to the current local transformation matrix ($L' = L \times T$). If the composition type is **PEXPostConcatenate**, the specified matrix is post-concatenated to the current local transformation matrix ($L' = T \times L$). If the composition type is **PEXReplace**, the specified matrix replaces the current local transformation matrix ($L' = T$).

The composite matrix is then recomputed using the current global transformation and the new local transformation matrix ($C = G \times L'$).

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetLocalTransform2D

PEXSetLocalTransform2D - Set Local Transformation 2D

Synopsis

void PEXSetLocalTransform2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *composition*, PEXMatrix3x3 *transform*)

Arguments

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**).
composition	The composition rule for combining the new matrix with the current local transform (**PEXPreConcatenate**, **PEXPostConcatenate**, **PEXReplace**).
transform	The new local transformation matrix.

Returns

None

Description

This function creates an output primitive attribute which modifies the local transformation matrix. This output command is similar to **PEXSetLocalTransform** except that the local transformation matrix is specified as a 3×3 matrix. Before modification of the local transformation matrix, the 3×3 matrix represented by

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

is expanded to a 4×4 matrix as follows:

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

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetLocalTransform

PEXSetMarkerBundleIndex - Set Marker Bundle Index

Synopsis

void PEXSetMarkerBundleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The marker bundle table index.

Returns

None

Description

This function creates an output primitive attribute which sets the marker bundle index attribute. If an undefined index is specified, the default bundle index one is used. If index one is undefined, the default values are used: marker type is **PEXMarkerAsterisk**; marker scale is 1.0 and marker color is indexed color one.

A marker bundle table index of 0 will produce a **BadPEXOutputCommand** error.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetMarkerColor - Set Marker Color

Synopsis

void PEXSetMarkerColor(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *color_type*, PEXColor **color*)

Arguments

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**).
color_type	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
color	A pointer to the marker color.

Returns

None

Description

This function creates an output primitive attribute which sets the marker color attribute. The attribute is set to either an indexed color or a direct color value, depending on the color type. If the color type is **PEXColorTypeIndexed** and the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value, e.g. the color type is **PEXColorTypeIndexed**, and the color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetMarkerColorIndex - Set Marker Color Index

Synopsis

void PEXSetMarkerColorIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The color table index for markers.

Returns

None

Description

This function creates an output primitive attribute which sets the marker color attribute to an indexed color with the value indicated by index. If the color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand	The color index value exceeds 65534.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetMarkerScale - Set Marker Scale

Synopsis

void PEXSetMarkerScale(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *scale*)

Arguments

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**).
scale	The marker scale factor.

Returns

None

Description

This function creates an output primitive attribute which sets the marker scale attribute. Scale is multiplied by the nominal marker size for the display device to produce a marker size mapped to the nearest supported marker size.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, **PEXSetIndividualASF**

PEXSetMarkerType - Set Marker Type

Synopsis

void PEXSetMarkerType(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *marker_type*)

Arguments

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**).
marker_type	The marker type (**PEXMarkerDot**, **PEXMarkerCross**, **PEXMarkerAsterisk**, **PEXMarkerCircle**, **PEXMarkerX**).

Returns

None

Description

This function creates an output primitive attribute which sets the marker type attribute. If the specified marker type is not supported, **PEXMarkerAsterisk** is used. Supported values for marker type are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetModelClipFlag - Set Model Clipping Flag

Synopsis

void PEXSetModelClipFlag(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *flag*)

Arguments

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**).
flag	The model clipping flag (**PEXClip** or **PEXNoClip**).

Returns

None

Description

This function creates an output primitive attribute which sets the model clip attribute. Values for the model clip flag are **PEXClip** (enable modeling clipping) and **PEXNoClip** (disable modeling clipping).

Errors

BadPEXOutputCommand	The output command contains an invalid value.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXSetModelClipVolume, **PEXSetModelClipVolume2D**, **PEXRestoreModelClipVolume**

PEXSetModelClipVolume - Set Model Clip Volume 3D

Synopsis

void PEXSetModelClipVolume(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *op*, unsigned int *count*, PEXHalfSpace **half_spaces*)

Arguments

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**).

op The model clipping volume operator (**PEXModelClipReplace** or **PEXModelClipIntersection**).

count The number of halfspaces.

half_spaces An array of points and normal vectors defining the model clipping volume.

Returns

None

Description

This function creates an output primitive attribute which sets the model clip volume attribute. The operator will be used to combine the specified list of half-spaces with the current model clipping volume to form a new model clipping volume. Values for the operator are **PEXModelClipReplace** (replace current volume) and **PEXModelClipIntersection** (intersect specified volume with current volume). Each half-space is defined by a point and a normal in model coordinates. The normal points in the direction of the half-space, and the point is considered to be on the plane. See **PEXGetImpDepConstants** for the maximum allowable number of model clip planes.

Each half-space is transformed by the current composite modeling transformation and combined with the current model clipping volume. The resulting model clipping volume is not affected by subsequent changes to the composite modeling transformation.

Data Structures

```
typedef struct {
    PEXCoord            point;
    PEXVector           vector;
} PEXHalfSpace;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetModelClipFlag, **PEXSetModelClipVolume2D**, **PEXRestoreModelClipVolume**, **PEXGetImpDepConstants**

PEXSetModelClipVolume2D - Set Model Clip Volume 2D

Synopsis

void PEXSetModelClipVolume2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *op*, unsigned int *count*, PEXHalfSpace2D **half_spaces*)

Arguments

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**).
op	The model clipping volume operator (**PEXModelClipReplace** or **PEXModelClipIntersection**).
count	The number of halfspaces.
half_spaces	An array of points and normal vectors defining the model clipping volume.

Returns

None

Description

This function creates an output primitive attribute which sets the model clip volume attribute. This output command is similar to **PEXSetModelClipVolume** except that the half-spaces are specified in 2D coordinates where the z-component of each point and normal vector assumed to be zero.

Data Structures

```
typedef struct {
    PEXCoord2D          point;
    PEXVector2D         vector;
} PEXHalfSpace2D;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetModelClipFlag, **PEXSetModelClipVolume**, **PEXRestoreModelClipVolume**, **PEXGetImpDepConstants**

PEXSetPatternAttributes - Set Pattern Interior Style Attributes

Synopsis

void PEXSetPatternAttributes(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord **ref_point*, PEXVector **vector1*, PEXVector **vector2*)

Arguments

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**).
ref_point	The pattern reference point.
vector1	The first pattern reference vector.
vector2	The second pattern reference vector.

Returns

None

Description

This function creates an output primitive attribute which sets the pattern reference point and both pattern reference vectors attributes, in modeling coordinates. If the surface interior style is **PEXInteriorStylePattern**, the pattern reference point and the pattern reference vectors are used to to position and scale the pattern on the surface. If either of the pattern reference vectors are zero length or if the vectors are parallel, the output command is ignored.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetPatternAttributes2D - Set Pattern Interior Style Reference Point

Synopsis

void PEXSetPatternAttributes2D(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXCoord2D **ref_point*)

Arguments

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**).
ref_point	The pattern reference point.

Returns

None

Description

This function creates an output primitive attribute which sets the pattern reference point and both pattern reference vector attributes, in 2D modeling coordinates. The pattern reference point is set to <x,y,0> and the two pattern reference vectors are set to <1,0,0> and <0,1,0>. If the surface interior style is **PEXInteriorStylePattern**, the pattern reference point and the reference vectors are used to position and scale the pattern on the surface.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetPatternSize - Set Pattern Interior Style Size

Synopsis

void PEXSetPatternSize(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *width*, double *height*)

Arguments

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**).
width	The pattern width.
height	The pattern height.

Returns

None

Description

This function creates an output primitive attribute which sets the pattern size attribute. Only the magnitude of the specified width and height is used. The value (width,0) is used as the pattern width vector, and the value (0,height) specifies the pattern height vector. If the interior style is **PEXInteriorStylePattern**, these values plus the pattern reference point and the pattern reference vector are used to position, scale, and rotate the pattern on the surface.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetParaSurfCharacteristics - Set Parametric Surface Characteristics

Synopsis

void PEXSetParaSurfCharacteristics(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *psc_type*, PEXPSCData **characteristics*)

Arguments

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**).
psc_type	The parametric surface characteristic type (**PEXPSCNone**, **PEXPSCImpDep**, **PEXPSCIsoCurves**, **PEXPSCMCLevelCurves**, **PEXPSCWCLevelCurves**).
characteristics	A pointer to data defining the parametric surface characteristics.

Returns

None

Description

This function creates an output primitive attribute which sets the parametric surface characteristics. If the specified parametric surface characteristics type is not supported, **PEXPSCNone** is used. Supported values for parameteric surface characteristic types are inquirable via **PEXGetEnumTypeInfo**.

Data Structures

```
typedef union {
    PEXPSCIsoparametricCurves iso_curves;
    PEXPSCLevelCurves        level_curves;
    PEXPSCImpDepData         imp_dep;
} PEXPSCData;

typedef struct {
    unsigned short      placement_type;
    unsigned short      reserved;
    unsigned short      u_count;
    unsigned short      v_count;
} PEXPSCIsoparametricCurves;

typedef struct {
    PEXCoord            origin;
    PEXVector           direction;
    unsigned short      count;                  /* number of parameters */
    unsigned short      reserved;
    float               *parameters;
} PEXPSCLevelCurves;

typedef struct {
    unsigned short      length;
    char                *data;
} PEXPSCImpDepData;
```

`See also the` *Common Data Structures* `section.`

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetPickID - Set Pick Identifier

Synopsis

void PEXSetPickID(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned long *pick_id*)

Arguments

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**).
pick_id	The pick identifier.

Returns

None

Description

This function creates an output primitive attribute which sets the current pick identifier attribute.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetPolylineInterpMethod - Set Polyline Interpolation Method

Synopsis

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

Arguments

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	The polyline interpolation method (**PEXPolylineInterpNone** or **PEXPolylineInterpColor**).

Returns

None

Description

This function creates an output primitive attribute which sets the polyline interpolation method attribute. If the specified interpolation method is not supported, **PEXPolylineInterpNone** is used. Supported values for polyline interpolation method are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetReflectionAttributes - Set Surface Reflection Attributes

Synopsis

void PEXSetReflectionAttributes(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXReflectionAttributes **attributes*)

Arguments

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**).
attributes	The surface reflection attributes.

Returns

None

Description

This function creates an output primitive attribute which sets the surface reflection attributes. Surface reflection attributes consist of the ambient, diffuse and specular coefficients, the specular concentration and color, and the transmission coefficient.

Data Structures

```
typedef struct {
    float               ambient;
    float               diffuse;
    float               specular;
    float               specular_conc;
    float               transmission;
    PEXColorSpecifier   specular_color;
} PEXReflectionAttributes;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXGetImpDepConstants**

PEXSetReflectionModel - Set Surface Reflection Model

Synopsis

void PEXSetReflectionModel(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *model*)

Arguments

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**).

model The surface reflection model (**PEXReflectionNone**, **PEXReflectionAmbient**, **PEXReflectionDiffuse**, **PEXReflectionSpecular**).

Returns

None

Description

This function creates an output primitive attribute which sets the surface reflection model attribute. If the specified reflection model is not supported, **PEXReflectionNone** is used. Supported values for surface reflection model are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetRenderingColorModel - Set Rendering Color Model

Synopsis

void PEXSetRenderingColorModel(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *model*)

Arguments

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**).
model	The rendering color model (**PEXRenderingColorModelImpDep**, **PEXRenderingColorModelRGB**, **PEXRenderingColorModelCIE**, **PEXRenderingColorModelHSV**, **PEXRenderingColorModelHLS**).

Returns

None

Description

This function creates an output primitive attribute which sets the rendering color model attribute. If the specified color model is not supported, an implementation-dependent model (model 0) is used. Supported values for rendering color model are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo

PEXSetSurfaceApprox - Set Surface Approximation Method

Synopsis

void PEXSetSurfaceApprox(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *method*, double *utolerance*, double *vtolerance*)

Arguments

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	The surface approximation method (**PEXApproxImpDep**, **PEXApproxConstantBetweenKnots**, **PEXApproxWCChordalSize**, **PEXApproxNPCChordalSize**, **PEXApproxDCChordalSize**, **PEXSurfaceApproxWCPlanarDev**, **PEXSurfaceApproxNPCPlanarDev**, **PEXSurfaceApproxDCPlanarDev**, **PEXApproxWCRelative**, **PEXApproxNPCRelative**, **PEXApproxDCRelative**).
utolerance	The surface approximation tolerance in the u direction.
vtolerance	The surface approximation tolerance in the v direction.

Returns

None

Description

This function creates an output primitive attribute which sets the surface approximation method attribute. If the specified method is not supported, **PEXApproxImpDep** is used. Supported values for surface approximation are inquirable via **PEXGetEnumTypeInfo**. The u and v tolerance values are provided to indicate the desired accuracy of the approximation, and are used in different ways for the different methods. See *Chapter 4: Pipeline Contexts* for more information about tolerance values.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXGetEnumTypeInfo**

PEXSetSurfaceColor - Set Surface Color

Synopsis

void PEXSetSurfaceColor(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *color_type*, PEXColor **color*)

Arguments

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**).

color_type The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).

color A pointer to the surface color.

Returns

None

Description

This function creates an output primitive attribute which sets the surface color attribute. The attribute is set to either an indexed color or a direct color value, depending on the color type. If the color type is **PEXColorTypeIndexed** and the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value, e.g. the color type is **PEXColorTypeIndexed**, and the color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetSurfaceColorIndex - Set Surface Color Index

Synopsis

void PEXSetSurfaceColorIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*,
unsigned int *index*)

Arguments

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**).
index	The color table index surfaces.

Returns

None

Description

This function creates an output primitive attribute which sets the surface color attribute to an indexed color with the value indicated by index. If the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand
The color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetSurfaceEdgeColor - Set Surface Edge Color

Synopsis

void PEXSetSurfaceEdgeColor(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*,
int *color_type*, PEXColor **color*)

Arguments

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**).
color_type	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
color	A pointer to the surface edge color.

Returns

None

Description

This function creates an output primitive attribute which sets the surface edge color attribute. The attribute is set to either an indexed color or a direct color value, depending on the color type. If the color type is **PEXColorTypeIndexed** and the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value, e.g. the color type is **PEXColorTypeIndexed**, and the color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetSurfaceEdgeColorIndex - Set Surface Edge Color Index

Synopsis

void PEXSetSurfaceEdgeColorIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The color table index for surface edges.

Returns

None

Description

This function creates an output primitive attribute which sets the surface edge color attribute to an indexed color with the value indicated by index. If the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand
The color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetSurfaceEdgeFlag - Set Surface Edge Flag

Synopsis

void PEXSetSurfaceEdgeFlag(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *flag*)

Arguments

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**).
flag	A flag indicating whether surface edge drawing is enabled or disabled (**PEXOn** or **PEXOff**).

Returns

None

Description

This function creates an output primitive attribute which sets the surface edge flag attribute. Values for the surface edge flag are **PEXOn** (enable surface edges) and **PEXOff** (disable surface edges).

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF

PEXSetSurfaceEdgeType - Set Surface Edge Type

Synopsis

void PEXSetSurfaceEdgeType(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *edge_type*)

Arguments

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**).
edge_type	The surface edge type (**PEXSurfaceEdgeSolid**, **PEXSurfaceEdgeDashed**, **PEXSurfaceEdgeDotted**, **PEXSurfaceEdgeDashDot**).

Returns

None

Description

This function creates an output primitive attribute which sets the surface edge type attribute. If the specified edge type is not supported, **PEXSurfaceEdgeSolid** is used. Supported values for surface edge type are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetSurfaceEdgeWidth - Set Surface Edge Width

Synopsis

void PEXSetSurfaceEdgeWidth(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, double *width*)

Arguments

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**).
width	The surface edge width scale factor.

Returns

None

Description

This function creates an output primitive attribute which sets the surface edge width attribute. The specified width is used as a scale factor. This scale factor is used to increase or decrease the width (in pixels) from the nominal edge width for the display device. The result is mapped to the nearest supported line width.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetImpDepConstants, **PEXSetIndividualASF**

PEXSetSurfaceInterpMethod - Set Surface Interpolation Method

Synopsis

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

Arguments

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	The surface interpolation method (**PEXSurfaceInterpNone**, **PEXSurfaceInterpColor**, **PEXSurfaceInterpDotProduct**, **PEXSurfaceInterpNormal**).

Returns

None

Description

This function creates an output primitive attribute which sets the surface interpolation method attribute. If the specified interpolation method is not supported, **PEXSurfaceInterpNone** is used. Supported values for surface interpolation are inquirable via **PEXGetEnumTypeInfo**.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXGetEnumTypeInfo, **PEXSetIndividualASF**

PEXSetTextAlignment - Set Text Alignment

Synopsis

void PEXSetTextAlignment(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *halignment*, int *valignment*)

Arguments

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**).
halignment	The horizontal text alignment (**PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignCenter**, **PEXHAlignRight**).
valignment	The vertical text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase**, **PEXVAlignBottom**).

Returns

None

Description

This function creates an output primitive attribute which sets the horizontal and vertical text alignment attributes.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextBundleIndex - Set Text Bundle Index

Synopsis

void PEXSetTextBundleIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*,
unsigned int *index*)

Arguments

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**).
index	The text bundle table index.

Returns

None

Description

This function creates an output primitive attribute which sets the text bundle index attribute. If an undefined index is specified, the default bundle index one is used. If index one is undefined, the defaults are used: text font index is one; text precision is **PEXStringPrecision**; character expansion is 1.0; character spacing is 0.0 and text color is indexed color one.

A text bundle table index of 0 will produce a **BadPEXOutputCommand** error.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetTextColor - Set Text Color

Synopsis

void PEXSetTextColor(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *color_type*, PEXColor **color*)

Arguments

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**).
color_type	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
color	A pointer to the text color.

Returns

None

Description

This function creates an output primitive attribute which sets the text color attribute. The attribute is set to either an indexed color or a direct color value, depending on the color type. If the color type is **PEXColorTypeIndexed** and the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Data Structures

See the *Common Data Structures* section.

Errors

BadPEXOutputCommand
The output command contains an invalid value, e.g. the color type is **PEXColorTypeIndexed**, and the color index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetTextColorIndex - Set Text Color Index

Synopsis

void PEXSetTextColorIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The color table index for text.

Returns

None

Description

This function creates an output primitive attribute which sets the text color attribute to an indexed color with the value indicated by index. If the specified color index is not defined, color index one is used. If color index one is not defined, the result is implementation-dependent.

Errors

BadPEXOutputCommand	The color index value exceeds 65534.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetTextFontIndex - Set Text Font Index

Synopsis

void PEXSetTextFontIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The text font table index.

Returns

None

Description

This function creates an output primitive attribute which sets the text font index attribute. Index specifies the entry in the current text font lookup table used to render text primitives. If index is undefined, the default index one is used. If index one is undefined, the result is implementation-dependent.

A text font table index less than 1 will produce a **BadPEXOutputCommand** error.

Errors

BadPEXOutputCommand	The output command contains an invalid value.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF, **PEXCreateLookupTable**

PEXSetTextPath - Set Text Path

Synopsis

void PEXSetTextPath(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *path*)

Arguments

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**).

path The text (drawing) path (**PEXPathRight**, **PEXPathLeft**, **PEXPathUp**, **PEXPathDown**).

Returns

None

Description

This function creates an output primitive attribute which sets the text path attribute.

Errors

BadPEXOutputCommand The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetTextPrecision - Set Text Precision

Synopsis

void PEXSetTextPrecision(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *precision*)

Arguments

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**).

precision The text precision (**PEXStringPrecision**, **PEXCharPrecision**, **PEXStrokePrecision**).

Returns

None

Description

This function creates an output primitive attribute which sets the text precision attribute.

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

If a character set value is not available in the current font group, then the entire string is rendered using the default font group. If a character set value is not available in the default font group, then that portion of the string is rendered in an implementation-dependent manner.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXSetIndividualASF

PEXSetViewIndex - Set View Index

Synopsis

void PEXSetViewIndex(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, unsigned int *index*)

Arguments

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**).
index	The view table index.

Returns

None

Description

This function creates an output primitive attribute which sets the view index attribute. If the specified view index is not defined, the default index zero is used. If view index zero is undefined, the default values are used: all clip flags are on; the clip limits are set to <0,0,0> , <1,1,1>; orientation and mapping are the identity matrices.

Errors

BadPEXOutputCommand
The view index value exceeds 65534.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateLookupTable

2.4. Structure Related and No-op Functions

There are some output commands that either relate to only structure resources or that do not directly cause anything to be rendered. The output commands in this category are application data, execute structure, generalized structure element, label and no-op. Label, no-op and application data have no visible effect when rendered. A label command is used to position the element pointer in a structure. For more information on labels, see *Chapter 7: Structures*.

2.4.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* similar to XErrorEvent - use to access additional info in OC
error */

      typedef struct {
          int             type;
          Display         *display;      /* Display the event was read from */
          XID             resourceid;    /* resource id of renderer or structure */
          unsigned long   serial;        /* serial number of failed request */
          unsigned char   error_code;    /* error code of failed request */
          unsigned char   request_code;  /* Major op-code of failed request */
          unsigned char   minor_code;    /* Minor op-code of failed request */
          unsigned short  op_code;       /* op-code of failed output command */
          unsigned short  count;         /* number of output commands successfully */
                                         /* executed before error */
      } PEXOCErrorEvent;
```

PEXApplicationData - Structure Application Data

Synopsis

void PEXApplicationData(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *length*, char **data*)

Arguments

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**).
length	The length, in bytes, of the application data.
data	A pointer to the application data.

Returns

None

Description

This function creates an application data output command which has no visible effect when rendered. It is typically used to store arbitrary application data in a structure.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXExecuteStructure - Execute Structure

Synopsis

void PEXExecuteStructure(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, PEXStructure *structure*)

Arguments

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**).
structure	The resource identifier of the structure.

Returns

None

Description

This function creates an execute structure output command which causes the specified structure to be executed during structure traversal. Executing a structure consists of these steps:

1. Save the current state of the rendering pipeline.
2. Set the global transform to the current composite modeling transform.
3. Set the local transform to the identity matrix.
4. Process all structure elements in the called structure.
5. Restore the state saved at step 1.

If structure does not exist at the time **PEXExecuteStructure** is processed, a **BadPEXOutputCommand** error is produced. The structure must first be created.

Errors

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateStructure, **PEXBeginStructure**, **PEXEndStructure**

PEXGSE - Generalized Structure Element

Synopsis

void PEXGSE(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, long *id*, int *length*, char **data*)

Arguments

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**).
id	The identifier of the GSE.
length	The length, in bytes, of the GSE data.
data	A pointer to the GSE data.

Returns

None

Description

This function creates a generalized structure element.

The complete interface and behavior for each GSE identifier should be available with the individual PEX server implementations. There are no standard PEX GSEs. If the specified GSE identifier is not supported, then the output command is ignored.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXLabel - Structure Label

Synopsis

void PEXLabel(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, long *label*)

Arguments

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**).
label	A value to be used as a label.

Returns

None

Description

This function creates a label output command which has no visible effect. Its primary function comes when it is stored in a structure, as labels can be used in various element pointer positioning operations. Labels within a particular structure need not be unique.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXDeleteToLabel, **PEXDeleteBetweenLabels**, **PEXSetElementPtrAtLabel**

PEXNoop - Noop Output Command

Synopsis

void PEXNoop(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*)

Arguments

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**).

Returns

None

Description

This function creates a no-op output command which, when rendered, will do nothing except update the current path.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

Chapter 3

Lookup Tables

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

Most attributes are normally obtained directly from the current state values of the rendering pipeline. Certain output primitive attributes (e.g., line color, marker type, surface interpolation method) have a related attribute, an aspect source flag, that indicates whether to obtain the attribute's value from the current pipeline state or from a specific entry in a lookup (bundle) table resource. Other attributes (e.g., viewing, depth-cueing parameters) are not stored as rendering pipeline state. Rather, these attributes are always obtained from a lookup table, and an index is stored as part of the rendering pipeline state to indicate the lookup table entry that is to be used.

Tables may be sparse (i.e. table indices need not be contiguous), therefore tables consist of index/entry pairs. The index is the numeric value that is used to access the data stored in an entry. The range of allowable indices for each type of table is given in the description of each table type. An entry is the collection of information (or the data record) that is defined for each type of table. The table descriptions in this section include the definition of an entry for each type of table. A table entry may contain one or more values, depending on the table type. For instance, in the **PEXLUTMarkerBundle** table type, each entry in the table consists of a marker type value, a marker scale value and a marker color value.

A lookup table may have predefined entries. These are entries that are filled in automatically by the server when the table is created. Predefined table entries may be overwritten. Each table also has a table index that is designated as the default index. If a specified table index is not defined, the contents of the default table index will be used instead. If the contents of the entry for the default table index are not defined, then the default attribute values will be used. The default index for all tables is index one, except for the view, depth cue, and color approximation tables, whose default index is index zero.

PEX lookup tables also support set versus realized semantics. A lookup table entry may be set to a value that is not exactly representable during rendering. This does not mean that a lookup table entry may be set to a value that is not supported by the implementation, such as enumerated types. Supported enumerated types are inquirable via **PEXGetEnumTypeInfo**. However, some supported enumerated types may not be realizable, and so may have to be approximated during rendering. This approximation is called the realized value. Such values will be silently mapped to an approximation during rendering. When retrieving entry values from a lookup table, applications may request the value as originally specified (the set value) or the value that is actually used when rendering (the realized value). In many cases, the set and realized values will be the same, but in some cases, they may not. All predefined entries are, by definition, realizable.

Types of Lookup Tables

The types of lookup tables defined are:

color approximation table
color table
depth cue table
edge bundle table
interior bundle table
light table
line bundle table
marker bundle table
pattern table
text bundle table
text font table

view table

Each of these types of lookup tables is described in detail in the following sections.

3.1. Color Approximation Lookup Tables

A color approximation table defines how a renderer will transform rendering pipeline color values into pixel values. An entry in a color approximation table may be defined using the **PEXColorApproxEntry** data type which consists of the following fields:

type - **PEXColorSpace** or **PEXColorRange**
model - **PEXColorApproxRGB**, **PEXColorApproxCIE**, **PEXColorApproxHSV**, **PEXColorApproxHLS**, or **PEXColorApproxYIQ**
max1 - maximum value for first component of converted color
max2 - maximum value for second component of converted color
max3 - maximum value for third component of converted color
mult1 - multiplier for positioning first component within computed pixel value
mult2 - multiplier for positioning second component within computed pixel value
mult3 - multiplier for positioning third component within computed pixel value
weight1 - weight value for first component for **PEXColorRange** type
weight2 - weight value for second component for **PEXColorRange** type
weight3 - weight value for third component for **PEXColorRange** type
base_pixel - value to add to computed pixel value
dither - hint as to whether to perform dithering

After a renderer has performed illumination, depth-cueing and clipping operations, it is left with a rendering pipeline color that must be converted to a displayable pixel value. The renderer's current color approximation index determines which entry in the color approximation table is used to perform the conversion. The rendering pipeline color value is first converted, only if necessary, to a color in the color space specified by the color model. The type of color approximation is then applied.

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

If the type is **PEXColorRange**, each component of the converted color from the rendering pipeline is multiplied by its corresponding weight value, after the weights have been normalized, and then the terms are added together to form a single value. The weight values can be adjusted to allow equal weighting of the components (weights are all equal) or to minimize or eliminate one or more of the components (one or more weights equal to 0). For instance, the weight values of 0.30, 0.59, 0.11 would convert an RGB value to a single-valued intensity according to the NTSC color standard.

Next, the computed value is multiplied by max1, which should be set to represent the number of color map entries in the range, minus 1. Max2 and max3 are not used. The value is then replicated for each of the three color components, which are then multiplied by mult1, mult2 and mult3 respectively, and then added together with the base pixel to form the final pixel value. For instance, if an application desires to display its computed image on a pseudo-color display using gray scale, it could allocate 100 contiguous color cells in the color map, and set max1 to the value 99, so that intensity values would be mapped into the range [0,99]. For visuals that use a single component to create a pixel value (e.g., PseudoColor), the application must specify the appropriate multipliers to select the appropriate channel (e.g. multipliers of 1.0, 0.0, and 0.0 could be used for a visual type of PseudoColor).

The use of addition rather than logical OR for composing pixel values permits allocations where the primary components are not allocated into distinct bit-planes. Since some hardware allows a performance improvement if the multiplication values are powers of two, **PEXGetImpDepConstants** can be called to get an indication of whether powers of two are preferred.

Dither is a hint as to whether some attempt at dithering should be performed. Whether dithering is supported (and the dithering algorithm that is used) are implementation-dependent. See **PEXGetImpDepConstants**.

The default index for color approximation tables is index zero, and index values may be in the range [0,65534], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.2. Color Lookup Tables

Color lookup tables (not to be confused with X color map resources) are used when dereferencing indexed color values. All color values in a color lookup table must therefore be specified as direct colors by using the **PEXColorEntry** data type which consists of the following fields:

> type - **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, or **PEXColorTypeRGB16**
> value - direct color value to be used for this index

The default index for color tables is one, and index values may be in the range [0,65534], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.3. Depth Cue Lookup Tables

Depth cue tables maintain depth-cueing information. An entry in a depth-cue table may be defined using the **PEXDepthCueEntry** data type which consists of the following fields:

> mode - **PEXOn** or **PEXOff**
> front_plane - the Z coordinate of the front reference plane
> back_plane - the Z coordinate of the back reference plane
> front_scaling - scaling factor to use at front reference plane
> back_scaling - scaling factor to use at back reference plane
> color - type and value of color to which others attenuate

The depth-cueing computation modifies colors according to the distance from the viewing position. The depth-cue color indicates the color to which objects will tend when going from the front plane to the back plane. The front and back scaling specify the values by which the colors will be scaled at the front and back depth-cueing planes. The depth-cueing computation is performed in NPC space -- the front plane is specified as the distance to the front depth-cueing plane in NPC coordinates and the back plane is specified as the distance to the back depth-cuing plane in NPC coordinates.

The default index for depth cue tables is zero, and index values may be in the range [0,65534], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.4. Edge Bundle Lookup Tables

The edge bundle lookup table maintains attributes for drawing edges of surface primitives. If the attribute source flag has been appropriately set, surface edge attributes may be obtained from the edge bundle table. An entry in an edge bundle table may be defined using the **PEXEdgeBundleEntry** data type which consists of the following fields:

> edge_flag - **PEXOn** or **PEXOff**
> type - **PEXSurfaceEdgeSolid**, **PEXSurfaceEdgeDashed**, **PEXSurfaceEdgeDotted**, or **PEXSurfaceEdgeDashDot**
> width - scale factor applied to the nominal width of surface edges
> color - type and value of color to be used when drawing surface edges

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

The default index for edge bundle tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.5. Interior Bundle Lookup Tables

The interior bundle table maintains attributes for drawing surface primitives. If the attribute source flag has been appropriately set, surface attributes may be obtained from the interior bundle table. An entry in an interior bundle table may be defined using the **PEXInteriorBundleEntry** data type which consists of the following fields:

- style - **PEXInteriorStyleHollow**, **PEXInteriorStyleSolid**, **PEXInteriorStylePattern**, **PEXInteriorStyleHatch**, or **PEXInteriorStyleEmpty**
- style_index - index to pattern or hatch style
- reflection_model - **PEXReflectionNone**, **PEXReflectionAmbient**, **PEXReflectionDiffuse**, or **PEXReflectionSpecular**
- interp_method - **PEXSurfaceInterpNone**, **PEXSurfaceInterpColor**, **PEXSurfaceInterpDotProduct**, or **PEXSurfaceInterpNormal**
- bf_style - same as style, but for back-facing surfaces
- bf_style_index - same as style_index, but for back-facing surfaces
- bf_reflection_model - same as reflection_model, but for back-facing surfaces
- bf_interp_method - same as interp_method, but for back-facing surfaces
- surface_approx - method and tolerances used to approximate surface primitives
- color - type and value of color to be used when drawing surface primitives
- reflection_attr - attributes to be used when computing reflection from surface primitives
- bf_color - same as color, but for back-facing surfaces
- bf_reflection_attr - same as reflection_attr, but for back-facing surfaces

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

The default index for interior bundle tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.6. Light Lookup Tables

A light table maintains light source definitions for use in reflectance calculations. An entry in a light table may be defined using the **PEXLightEntry** data type which consists of the following fields:

- type - **PEXLightAmbient**, **PEXLightWCVector**, **PEXLightWCPoint**, or **PEXLightWCSpot**
- direction - direction of light source (a unit length vector which points away from the light)
- point - position of light source
- concentration - concentration exponent
- spread_angle - half-angle
- attenuation1 - light attenuation factor 1
- attenuation2 - light attenuation factor 2
- color - type and value of color of light source

Light type must be one of the constants **PEXLightAmbient**, **PEXLightWCVector**, **PEXLightWCPoint**, **PEXLightWCSpot**. A server implementation may not support all light types, and may define some new ones. (See **PEXGetEnumTypeInfo**.) Depending upon the type of light, some of the values in a light table entry may be ignored.

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

A world coordinate vector light source (or directional light source) is defined by direction and color. This type of light is located at infinity so all light rays which emanate from it are parallel when they strike an object's surface.

A world coordinate point light source (or positional light source) is defined by color, a position in world coordinates, and attenuation. Positional lights illuminate equally in all directions. The attenuation indicates how the intensity of the point light source drops off as a function of distance.

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

There may be a limit to the number of non-ambient lights that can be enabled at one time. See **PEXGetImpDepConstants.**

The default index for light tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.7. Line Bundle Lookup Tables

Line bundle tables maintain attributes for drawing polyline and curve primitives. If the attribute source flag has been appropriately set, polyline and curve attributes may be obtained from a line bundle table. An entry in a line bundle table may be defined using the **PEXLineBundleEntry** data type which consists of the following fields:

> type - **PEXLineTypeSolid**, **PEXLineTypeDashed**, **PEXLineTypeDotted**, or **PEXLineTypeDashDot**
> interp_method - **PEXPolylineInterpNone** or **PEXPolylineInterpColor**
> curve_approx - approximation method and tolerance to be used to draw curve primitives
> width - scale factor applied to the nominal width of lines and curves
> color - type and value of color to be used when drawing lines and curves

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

The default index for line bundle tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.8. Marker Bundle Lookup Tables

Marker bundle tables maintain attributes for drawing marker primitives. If the attribute source flag has been appropriately set, marker attributes may be obtained from a marker bundle table. An entry in a marker bundle table may be defined using the **PEXMarkerBundleEntry** data type which consists of the following fields:

> type - **PEXMarkerDot**, **PEXMarkerCross**, **PEXMarkerAsterisk**, **PEXMarkerCircle**, or **PEXMarkerX**
> scale - scale factor to be applied to marker
> color - type and value of color to be used to draw markers

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

The default index for marker bundle tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.9. Pattern Lookup Tables

An entry in a pattern table may be defined using the **PEXPatternEntry** data type which consists of the following fields:

> color_type - type of color values
> row_count - number of rows in the colors array
> col_count - number of columns in the colors array
> colors - an array of colors defining the pattern

The pattern table defines the available patterns for use when the pipeline interior style or back-facing interior style attribute is **PEXInteriorStylePattern**. interior style index attributes specify the table index to be used. The colors are stored row-major with the upper-left hand cell being the first in the list of colors followed by the remaining cells in the first row, then the second row, and so on.

The default index for pattern tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.10. Text Bundle Lookup Tables

The text bundle lookup table maintains attributes for drawing text and annotation text primitives. If the attribute source flag has been appropriately set, text and annotation text attributes may be obtained from a text bundle lookup table. An entry in a text bundle table may be defined using the **PEXTextBundleEntry** data type which consists of the following fields:

font_index - index of text font to be used
precision - **PEXStringPrecision**, **PEXCharPrecision**, or **PEXStrokePrecision**
char_expansion - horizontal character scaling factor
char_spacing - spacing to be used between characters
color - type and value of color to be used to draw text

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

The default index for text bundle tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.11. Text Font Lookup Tables

Text font tables maintain a list of PEX and X11 font groups. Font groups are selected using the text font index. Inside a font group a specific font is selected by the character set attribute of a mono encoding. (A mono encoding is a portion of a text string.) An entry in a text font lookup table may be defined using a **PEXTextFontEntry** data type which consists of the following fields:

count - the number of PEX fonts in the fonts array
fonts - a pointer to the array of font identifiers

The default index for text font tables is one, and index values may be in the range [1,65535], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

3.12. View Lookup Tables

A view table maintains viewing information. "Views" are specified as indices, which are used to identify the appropriate information in a view table. An entry in the view table may be defined using the **PEXViewEntry** data type which consists of the following fields:

clip_flags - a mask made up of **PEXClipXY**, **PEXClipBack**, **PEXClipFront**, **PEXClippingAll**
clip_limits - endpoints of diagonal that defines clipping volume
orientation - view orientation transformation matrix
mapping - view mapping transformation matrix

The clip limits specify the minimum and maximum values of a rectangular volume in normalized projection coordinates. Clip flags contains three flags that indicate whether to clip against the side, back, and front planes of the volume specified by clip limits. Use the constants **PEXClipXY**, **PEXClipBack**, **PEXClipFront**, or **PEXClippingAll** (which sets all three clipping flags) for use in constructing the value for clip flags. The actual clipping volume for each view is a rectangular parallelpiped that is computed by taking the intersection of the view clipping volume (determined by the clip limits and the clip flags) and the NPC subvolume.

Orientation is a transformation matrix that maps geometry in world coordinates to viewing reference coordinates (also known as viewing coordinates). Mapping is a transformation matrix that maps geometry in view reference coordinates to normalized projection coordinates. The mapping transformation is typically a parallel or perspective projection. Both of these matrices are 4-by-4 row-major matrices. All coordinate systems in PEX are right-handed coordinate systems.

The default index for view tables is zero, and index values may be in the range [0,65534], of which there are n definable entries (see **PEXGetTableInfo** to obtain the value of n).

PEXCopyLookupTable - Copy Lookup Table

Synopsis

void PEXCopyLookupTable(Display **display*, PEXLookupTable *src_table*, PEXLookupTable *dst_table*)

Arguments

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

src_table The resource identifier of the source lookup table.

dst_table The resource identifier of the destination lookup table.

Returns

None

Description

This function copies entries from the source lookup table to destination lookup table, after first deleting all entries in the destination lookup table. Both tables must already exist as valid lookup table resources. Both must have been created for the same class of drawables, and both must be the same table type.

Data Structures

```
typedef XID              PEXLookupTable;
```

Errors

BadMatch The specified lookup tables were not created with compatible drawables.

BadPEXLookupTable
A specified lookup table resource identifier is invalid, or the table type is unsupported.

See Also

PEXCreateLookupTable

PEXCreateLookupTable - Create Lookup Table

Synopsis

PEXLookupTable PEXCreateLookupTable(Display **display*, Drawable *drawable*, int *table_type*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
table_type	The type of lookup table to be created (see the Description).

Returns

The resource identifier of the newly-created lookup table.

Description

This function creates a lookup table resource of the specified type and returns the resource identifier of the new lookup table. The returned identifier is used to refer to the created lookup table. The type of lookup table to be created must be one of the following:

PEXLUTColorApprox
PEXLUTColor
PEXLUTDepthCue
PEXLUTEdgeBundle
PEXLUTInteriorBundle
PEXLUTLight
PEXLUTLineBundle
PEXLUTMarkerBundle
PEXLUTPattern
PEXLUTTextBundle
PEXLUTTextFont
PEXLUTView

The newly-created lookup table can only be used with drawables having the same depth and root as the specified drawable.

Data Structures

```
typedef XID             PEXLookupTable;
```

Errors

BadAlloc	The server failed to allocate the resource.
BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported.
BadPEXLookupTable	The specified table type is unsupported.
BadValue	The specified table type is invalid.

See Also

PEXFreeLookupTable

PEXDeleteTableEntries - Delete Lookup Table Entries

Synopsis

void PEXDeleteTableEntries(Display **display*, PEXLookupTable *table*, unsigned int *start*, unsigned int *count*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
table	The resource identifier of the lookup table.
start	The initial table entry to be deleted.
count	The number of entries to be deleted.

Returns

None

Description

This function deletes lookup table entries from the specified lookup table, starting at the specified entry. Entries with index values between the start and (start + count-1), inclusive, are deleted. Attempts to delete undefined entries are ignored.

Data Structures

```
typedef XID                PEXLookupTable;
```

Errors

BadPEXLookupTable
: The specified lookup table resource identifier is invalid, or the table type is unsupported.

BadValue
: The sum of start and count is too large, or index 0 is invalid for the specified table type.

See Also

PEXGetTableInfo, **PEXGetPredefinedEntries**, **PEXGetDefinedIndices**

PEXFreeLookupTable - Free Lookup Table

Synopsis

void PEXFreeLookupTable(Display **display*, PEXLookupTable *table*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
table	The resource identifier of the lookup table.

Returns

None

Description

This function deletes the association between the lookup table resource identifier and the lookup table. The lookup table is freed when no other resource references it.

Errors

BadPEXLookupTable
The specified lookup table resource identifier is invalid.

See Also

PEXCreateLookupTable

PEXFreeTableEntries - Free Storage associated with Tables Entries Return Parameters

Synopsis

void PEXFreeTableEntries(int *table_type*, unsigned int *count*, PEXPointer *entries*)

Arguments

table_type	The type of table entries in the array.
count	The number of entries in the array.
entries	An array of table entries.

Returns

None

Description

This function deallocates memory returned by **PEXGetPredefinedEntries**, **PEXGetTableEntries**, **PEXGetTableEntry**

Errors

None

See Also

PEXGetPredefinedEntries, **PEXGetTableEntries**, **PEXGetTableEntry**

PEXGetDefinedIndices - Get Lookup Table Defined Indices

Synopsis

Status PEXGetDefinedIndices(Display **display*, PEXLookupTable *table*, unsigned long **count_return*, PEXTableIndex ***indices_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
table	The resource identifier of the lookup table.
count_return	Returns the number of returned indices.
indices_return	Returns an array of indices to defined table entries.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns the defined indices for the specified lookup table. The index of each defined table entry is returned in a list.

PEXlib allocates memory for the returned indices. **XFree** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXLookupTable;
typedef unsigned short  PEXTableIndex;
```

Errors

BadPEXLookupTable

The specified lookup table resource identifier is invalid, or the table type is unsupported.

See Also

PEXCreateLookupTable, **PEXGetTableInfo**

PEXGetPredefinedEntries - Get Lookup Table Predefined Indices

Synopsis

Status PEXGetPredefinedEntries(Display **display*, Drawable *drawable*, int *table_type*, unsigned int *start*, unsigned int *count*, PEXPointer **entries_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
table_type	The type of lookup table (see the Description).
start	The index of the first predefined entry to be returned.
count	The number of predefined entries requested.
entries_return	Returns an array of predefined table entries.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns the predefined entries for the specified lookup table. The type must be one of the following:

PEXLUTColorApprox
PEXLUTColor
PEXLUTDepthCue
PEXLUTEdgeBundle
PEXLUTInteriorBundle
PEXLUTLight
PEXLUTLineBundle
PEXLUTMarkerBundle
PEXLUTPattern
PEXLUTTextBundle
PEXLUTTextFont
PEXLUTView

Predefined entries are those automatically filled with valid data when a lookup table is created. The query is conducted based on the assumption that the lookup table would be used on drawables with the same root and depth as the specified drawable. The function returns at most the specified number of entries, starting with the specified entry index. The number of entries requested can not be larger than the number of predefined entries. Call **PEXGetTableInfo** to determine the number of predefined entries.

Entries in the returned list depend on the type specified. See **PEXSetTableEntries** for the structure types.

PEXlib allocates the memory for the returned table entries. **PEXFreeTableEntries** should be called to deallocate the memory.

Data Structures

```
#if NeedFunctionPrototypes
typedef void *PEXPointer;
#else
typedef char *PEXPointer;
#endif
```

Errors

BadDrawable The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported.

BadValue The specified table type is invalid or unsupported, start is less than the minimum predefined entry, the sum of start and count is greater than the maximum predefined entry, or index 0 is not valid for the specified table type.

BadPEXLookupTable
The specified table type is unsupported.

See Also

PEXCreateLookupTable, **PEXGetTableInfo**, **PEXGetDefinedIndices**, **PEXGetTableEntry**, **PEXGetTableEntries**, **PEXSetTableEntries**

PEXGetTableEntries - Get Lookup Table Entries

Synopsis

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

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
table	The resource identifier of the lookup table from which the table entries are to be obtained.
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 a table entry in the requested range is not defined, the default entry for a table of this type is returned.

The type of entries in the returned list depend on the returned table type. See **PEXSetTableEntries** for the structure types.

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 the memory for the returned entries. **PEXFreeTableEntries** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXLookupTable;

#if NeedFunctionPrototypes
typedef void *PEXPointer;
#else
typedef char *PEXPointer;
#endif
```

Errors

BadPEXLookupTable
The specified lookup table resource identifier is invalid, or the table type is unsupported.

BadValue The sum of start and count is too large, or index 0 is invalid for the specified table type.

See Also

PEXCreateLookupTable, **PEXGetTableInfo**, **PEXGetPredefinedEntries**, **PEXGetDefinedIndices**, **PEXGetTableEntry**, **PEXSetTableEntries**

PEXGetTableEntry - Get Lookup Table Entry

Synopsis

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

Arguments

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, either **PEXDefinedEntry** if the specified lookup table entry is defined or **PEXDefaultEntry** if it is undefined.
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. See **PEXSetTableEntries** for the structure types.

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 deallocate the memory.

Data Structures

```
typedef XID             PEXLookupTable;

#if NeedFunctionPrototypes
typedef void *PEXPointer;
#else
typedef char *PEXPointer;
#endif
```

Errors

BadPEXLookupTable
: The specified lookup table resource identifier is invalid, or the table type is unsupported.

BadValue
: Index 0 is invalid for the specified table type.

See Also

PEXCreateLookupTable, **PEXGetTableInfo**, **PEXGetPredefinedEntries**, **PEXGetDefinedIndices**, **PEXGetTableEntries**, **PEXSetTableEntries**

PEXGetTableInfo - Get Lookup Table Information

Synopsis

Status PEXGetTableInfo(Display **display*, Drawable *drawable*, int *table_type*, PEXTableInfo **info_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
table_type	The type of lookup table (see the Description).
info_return	Returns the lookup table information.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns information about the specified type of lookup table. The type of lookup table must be one of the following:

PEXLUTColorApprox
PEXLUTColor
PEXLUTDepthCue
PEXLUTEdgeBundle
PEXLUTInteriorBundle
PEXLUTLight
PEXLUTLineBundle
PEXLUTMarkerBundle
PEXLUTPattern
PEXLUTTextBundle
PEXLUTTextFont
PEXLUTView

The returned information is based on the assumption that the lookup table would be used on drawables with the same root and depth as specified drawable. The returned information includes the number of predefined table entries, the number of definable table entries, and the indices of the predefined minimum and maximum entries. Predefined entries are contiguous. The number of definable table entries includes the number of predefined entries since predefined table entries can be redefined by the application. All entries between the predefined minimum and maximum values are guaranteed to be defined initially (i.e. predefined entries have contiguous indices).

Data Structures

```
typedef struct {
    unsigned short      definable_entries;
    unsigned short      predefined_count;
    unsigned short      predefined_min;
    unsigned short      predefined_max;
} PEXTableInfo;
```

Errors

BadDrawable	The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported.

BadPEXLookupTable
The specified table type is unsupported.

BadValue The specified table type is invalid.

See Also

PEXCreateLookupTable

PEXSetTableEntries - Set Lookup Table Entries

Synopsis

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

Arguments

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 table entries to be set.
type	The type of lookup table entries to be set (see the Description).
entries	An array of table entries.

Returns

None

Description

This function sets lookup table entries in the specified 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 type is **PEXLUTTextFont**
PEXViewEntry if type is **PEXLUTView**
PEXColorApproxEntry if type is **PEXLUTColorApprox**
PEXLineBundleEntry if type is **PEXLUTLineBundle**
PEXMarkerBundleEntry if type is **PEXLUTMarkerBundle**
PEXTextBundleEntry if type is **PEXLUTTextBundle**
PEXInteriorBundleEntry if type is **PEXLUTInteriorBundle**
PEXEdgeBundleEntry if type is **PEXLUTEdgeBundle**
PEXLightEntry if type is **PEXLUTLight**
PEXDepthCueEntry if type is **PEXLUTDepthCue**
PEXColorEntry if type is **PEXLUTColor**
PEXPatternEntry if type is **PEXLUTPattern**

Data Structures

```
typedef XID             PEXLookupTable;

#if NeedFunctionPrototypes
typedef void *PEXPointer;
#else
typedef char *PEXPointer;
#endif
```

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, or the table type is

unsupported.

BadValue The sum of start plus count is too large, a table entry field contains an invalid value, or index 0 is invalid for the specified table type.

See Also

PEXCreateLookupTable, **PEXGetTableInfo**, **PEXGetPredefinedEntries**, **PEXGetDefinedIndices**, **PEXGetTableEntry**, **PEXGetTableEntries**

Chapter 4

Pipeline Contexts

A pipeline context is a PEX resource containing an instance of the attributes which describe a rendering pipeline. The attributes in a pipeline context are used to initialize the renderer resource's pipeline state whenever a **PEXBeginPickAll**, **PEXBeginPickOne**, **PEXBeginRendering**, **PEXPickAll**, **PEXPickOne** or **PEXRenderNetwork** function is called. The pipeline context section describes pipeline context attributes and the operations that can be performed on pipeline context resources.

Some of the attributes in a pipeline context are references to other resources. If a resource identifier is used in creating another resource, but then that first resource is freed, the value **PEXAlreadyFreed** is returned as the resource identifier of the first resource when the later resource is queried.

The following sections describe the attributes of a pipeline context and how attributes affect the operation of the rendering pipeline.

In addition to the descriptions of attributes given here, see also *Chapter 2: Output Commands*.

4.1. Marker Attributes

The attributes described in this section control the appearance of marker primitives.

The marker type attribute specifies the marker type (glyph) to use when drawing marker primitives. The supported marker types are inquirable via **PEXGetEnumTypeInfo**. The **PEXMarkerDot** type displays the smallest displayable dot at the marker position. The marker scale factor is ignored for **PEXMarker-Dot**. The **PEXMarkerCross** type displays a cross or plus sign with the intersection at the marker position. The **PEXMarkerAsterisk** type displays a star or asterisk with the center at the marker position. The **PEXMarkerCircle** type displays a circle with the center at the marker position. The **PEXMarkerX** type displays an "x" or diagonal cross with the intersection at the marker position.

The default marker type is **PEXMarkerAsterisk**.

The marker scale attribute contains the marker scale factor to use when drawing marker primitives. It is multiplied by the nominal marker size (see **PEXGetImpDepConstants**) to determine the actual size of the marker to be drawn.

The default marker scale is 1.0.

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

The default marker color has color type **PEXColorTypeIndexed** and a color value of one.

The marker bundle index attribute contains the lookup table index used to obtain bundled marker attributes from the marker bundle table.

The default marker bundle index is one.

4.2. Text Attributes

The attributes described in this section control the appearance of text and annotation text primitives. Text and annotation text primitives share the text font index, text precision, character expansion, character spacing, text color, and text bundle index attributes. Text and annotation text primitives use separate path, alignment, height, and up vector attributes.

The text font index attribute contains the lookup table index used to obtain the font identification for drawing text and annotation text primitives.

The default text font index is one.

The text precision attribute contains the text precision to use when drawing text and annotation text primitives. The possible values are **PEXStrokePrecision**, **PEXCharPrecision** and **PEXStringPrecision**. For **PEXStrokePrecision** precision, clipping and transformation occurs on every stroke that defines a character glyph. For **PEXCharPrecision** precision, clipping and transformation occurs on at least a character-by-character basis. A text string in **PEXStringPrecision** precision is treated as a single entity that is transformed and clipped.

The default text precision is **PEXStringPrecision**.

The character expansion attribute specifies the deviation of the width to height ratio of the characters from the ratio indicated by the font designer.

The default character expansion is 1.0.

The character spacing attribute specifies how much additional space is to be inserted between two adjacent character bodies and is specified as a fraction of the font-nominal character height.

The default character spacing is 0.0.

The text color attribute contains the color value to use when drawing text primitives.

The default text color has color type **PEXColorTypeIndexed** and a color value of one.

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

The default character height is 0.01.

The character up vector attribute contains the character up vector to use when drawing text primitives. The up vector is specified in the text local coordinate system defined by the direction vectors specified with the text primitive.

The default character up vector is <0.0, 1.0>.

The text path attribute contains the text path to use when drawing text primitives (i.e., the writing path of the text string). Possible values for text path are **PEXPathLeft** (text path is to the left), **PEXPathRight** (text path is to the right), **PEXPathUp** (text path is up), and **PEXPathDown** (text path is down).

The default text path is **PEXPathRight**.

The text alignment attribute contains the horizontal and vertical alignment to use when drawing text primitives. The possible values for horizontal alignment are **PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignRight**, and **PEXHAlignCenter**. If horizontal alignment is **PEXHAlignNormal**, text string alignment is "normal" with respect to the current text path. A text string with **PEXHAlignLeft** horizontal alignment is left-justified, a text string with **PEXHAlignRight** horizontal alignment is right-justified, and a text string with **PEXHAlignCenter** horizontal alignment is center-justified. The possible values for vertical alignment are **PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase**, and **PEXVAlignBottom**. If vertical alignment is **PEXVAlignNormal**, text string alignment is "normal" with respect to the current text path. Text strings with **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase** and **PEXVAlignBottom** vertical alignment are aligned with the character top line, cap line, half line, base line and bottom line, respectively.

The default text horizontal alignment is **PEXHAlignNormal** and the default text vertical alignment is **PEXVAlignNormal**.

The annotation text height attribute has the same meaning for annotation text primitives as the text height attribute has for text primitives.

The default annotation text height is 0.01.

The annotation text up vector attribute has the same meaning for annotation text primitives as the text up vector attribute has for text primitives.

The default annotation text up vector is <0.0, 1.0>.

The annotation text path attribute has the same meaning for annotation text primitives as the text path attribute has for text primitives.

The default annotation text path is **PEXPathRight**.

The annotation text alignment attribute has the same meaning for annotation text primitives as the text alignment attribute has for text primitives.

The default annotation text horizontal alignment is **PEXHAlignNormal** and the default annotation text vertical alignment is **PEXVAlignNormal**.

The annotation text style attribute specifies the style to use when drawing annotation text primitives. Supported annotation text styles are inquirable via **PEXGetEnumTypeInfo**. If the style is **PEXATextConnected** a line is drawn, using the current line attributes, from the annotation text reference point to the annotation text string. If the style is **PEXATextNotConnected**, no connecting line is drawn.

The default annotation text style is **PEXATextNotConnected**.

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

The default text bundle index is one.

4.3. Line and Curve Attributes

The attributes in this section control the appearance of polylines, non-uniform rational B-spline curves. Also, when the annotation text style is **PEXATextConnected**, the connection lines are drawn using the current set of line and curve attributes.

The line type attribute specifies the style to use when drawing line or curve primitives. Supported line types are inquirable via **PEXGetEnumTypeInfo**. A line drawn with a line type of **PEXLineTypeSolid** is drawn as a solid, unbroken line. A line of type **PEXLineTypeDashed** is drawn using dashed lines. A line of type **PEXLineTypeDotted** is drawn using dotted lines. A line of type **PEXLineTypeDashDot** is drawn with alternating dots and dashes.

The default line type is **PEXLineTypeSolid**.

The line width attribute is the scale factor applied to the nominal width of the polyline or curve primitive to be rendered. Line width is applied in 2D device coordinates after the primitive has been transformed from 3D space to 2D device space. It is multiplied by the nominal line width (see **PEXGetImpDepConstants**) to determine the actual width of the line or curve to be drawn.

The default line width scale factor is 1.0.

The line color attribute contains the color value to use when drawing line or curve primitives.

The default line color has color type **PEXColorTypeIndexed** and a color value of one.

The curve approximation attribute affects only non-uniform rational B-spline curves. Supported curve approximation methods are inquirable via **PEXGetEnumTypeInfo**. If the curve approximation method is **PEXApproxImpDep**, an implementation-dependent method is used to approximate the curve. The tolerance value is not used for this method. If the method is **PEXApproxConstantBetweenKnots**, the curve is tessellated with equal parametric increments between successive pairs of knots. When the curve includes knots and the tolerance is negative, the curve is evaluated at just the parametric bounds, plus all knots which are between the parametric bounds. If the tolerance is positive, the curve is evaluated at the number of points between the knots, but only if these parametric values are between the parametric bounds. If the method is **PEXApproxWCChordalSize**, the curve is tessellated until the length of every line segment, in world coordinates, is less than the tolerance. If the method is **PEXApproxNPCChordalSize**, the curve is tessellated until the length of every line segment, in normalized projection coordinates, is less than the tolerance. If the method is **PEXApproxDCChordalSize**, the curve is tessellated until the length of every line segment, in device coordinates, is less than the tolerance. If the method is **PEXCurveApproxWCChordalDev**, the curve is tessellated until the maximum deviation, in world coordinates, between the line and the curve is less than the tolerance. If the method is **PEXCurveApproxNPCChordalDev**, the curve is tessellated until the maximum deviation, in normalized projection coordinates, between the line and the curve is less than the tolerance. If the method is **PEXCurveApproxDCChordalDev**, the curve is tessellated until the maximum deviation, in device coordinates, between the line and the curve is less than the tolerance. If the method is **PEXApproxWCRelative**, the tolerance value must be between 0 and 1. The tolerance specifies an approximation value which indicates a relative quality of rendering to be maintained independent of scaling in world coordinates. Values closer to 1 specify a better relative quality. If the method is **PEXApproxNPCRelative**, the tolerance value must be between 0 and 1. The tolerance specifies an approximation value which indicates a relative quality of rendering to be maintained independent of scaling in normalized projection coordinates. Values closer to 1 specify a better relative quality. If the method is **PEXApproxDCRelative**, the tolerance value must be between 0 and 1. The tolerance specifies an approximation value which indicates a relative quality of rendering to be maintained independent of scaling in device coordinates. Values closer to 1 specify a better relative quality.

The default curve approximation method is **PEXApproxImpDep**.

The polyline interpolation attribute does not affect non-uniform rational B-spline curves. Supported polyline interpolation methods are inquirable via **PEXGetEnumTypeInfo**. If the method is **PEXPolylineInterpNone**, color values are not interpolated between vertices, when they are provided. If colors are specified at the vertices of a polyline set primitive, the color of the ith vertex is used when drawing the line segment between the ith and i+1th vertices. If the method is **PEXPolylineInterpColor**, color values are interpolated along the line segment for pixels between the vertices.

The default polyline interpolation method is **PEXPolylineInterpNone**.

The line bundle index attribute contains the lookup table index used to obtain bundled line and curve attributes from the line bundle table.

The default line bundle index is one.

4.4. Surface Attributes

This section describes attributes that affect the appearance of area-defining (surface) primitives. The list of surface primitives includes fill areas, fill area sets, triangle strips, quadrilateral meshes, non-uniform rational B-spline surfaces and SOFAS (Set Of Fill Area Sets).

The surface interior style attribute defines the interior style used when drawing front-facing surfaces. This attribute will also be used as the interior style for back-facing surfaces if the surface distinguish attribute is **False**. Supported surface interior styles are inquirable via **PEXGetEnumTypeInfo**. If the surface interior style is **PEXInteriorStyleHollow**, only the boundary is drawn. If the interior style is **PEXInteriorStyleSolid**, the surface interior is filled. If the interior style is **PEXInteriorStylePattern**, the surface interior is filled with the currently selected pattern, oriented according to the current pattern size, pattern reference point, and pattern reference vectors. If the interior style is **PEXInteriorStyleHatch**, the surface interior is filled with the currently selected hatch style. If the interior style is **PEXInteriorStyleEmpty**, the surface interior is not drawn at all.

The default interior style is **PEXInteriorStyleHollow**.

If the current surface interior style is **PEXInteriorStylePattern**, the surface interior style index attribute selects which pattern will be used to render fill areas. If the current surface interior style is **PEXInteriorStyleHatch**, this index selects which hatch style will be used to render fill areas. This attribute is not used with other interior styles.

The default interior style index is one.

The surface color attribute contains the color value to use when drawing surfaces (provided the primitive does not have facet or vertex colors which override the surface color) that are front-facing with respect to the viewing position. It will also be used as the color for back-facing surfaces if the surface distinguish attribute is **False**.

The default front-facing surface color has color type **PEXColorTypeIndexed** and a color value of one.

The surface reflection attributes are used during the reflectance calculation stage of the rendering pipeline. These attributes will also be used as the surface reflection attributes for back-facing primitives if the surface distinguish attribute is **False**. The specular color attribute provides an additional coefficient per primary for use in the specular reflection computation. This allows highlights to be computed that are some color other than that of the light source. The specular concentration defines the sharpness of the specular highlights or the "shininess" of the surface. This value is typically used as the exponent in the specular reflection term of lighting equations and it ranges from zero to the maximum floating point value. If the specular concentration is zero, specular highlights are very broad. If the specular concentration is much larger than zero, the highlights are very small and sharp, as if the surface was very shiny. The transmission coefficient indicates the amount of light that passes through a surface. A transmission coefficient of zero indicates the surface is opaque (lets no light through); a transmission coefficient of 1.0 indicates the surface is totally invisible (lets all light through).

The default surface reflection attributes are: ambient coefficient = 1.0, diffuse coefficient = 1.0, specular coefficient = 1.0, specular concentration = 0.0, transmission coefficient = 0.0, specular color type = **PEXColorTypeIndexed** and color value of one.

The surface reflection model attribute defines the method to use in approximating the effect of light reflecting from surface primitives. This attribute will also be used for back-facing primitives if the surface distinguishing attribute is **False**. Supported surface reflection models are inquirable via **PEXGetEnumTypeInfo**. If the surface reflection model is **PEXReflectionNone**, no reflectance calculation is performed and the surface color is not affected by lighting. If the reflection model is **PEXReflectionAmbient**, only the ambient terms of the lighting equation are used. The shaded color will be the object's base color as seen under ambient light. If the reflection model is **PEXReflectionDiffuse**, only the ambient and diffuse terms of the lighting equation are used. The shaded color will be the base color as seen under ambient light, plus a diffuse reflection component from each light source. If the reflection model is **PEXReflectionSpecular**, the ambient, diffuse, and specular terms of the lighting equation are all used during the reflectance calculation. The shaded color will be the same as for **PEXReflectionDiffuse**, plus a specular reflection component from each light source.

The default surface reflection model is **PEXReflectionNone**.

The surface interpolation attribute specifies the method used to compute color values in surface interiors when rendering surface primitives. This attribute will also be used for back-facing primitives if the surface distinguish attribute is **False**. Supported surface interpolation methods are inquirable via **PEXGetEnumTypeInfo**. If the surface interpolation method is **PEXSurfaceInterpNone**, the color resulting from a single reflectance calculation is used for the entire surface. No interpolation will be performed across surface interiors or edges. This surface interpolation method is also known as "flat-shading." If the method is **PEXSurfaceInterpColor**, the colors are computed at the vertices of the surface according to the current reflection model. These color values are then linearly interpolated across the interior of the surface or the edges. If the method is **PEXSurfaceInterpDotProduct**, the lighting equation dot products are computed at the vertices. These dot products are linearly interpolated and shading is applied using these values to compute the color value at each pixel in the interior of a surface or along a surface edge. If the method is **PEXSurfaceInterpNormal**, the reflectance calculation is performed at each pixel in the surface interior or along a surface edge using the interpolated normal.

The default surface interpolation method is **PEXSurfaceInterpNone**.

If the distinguish attribute is **True**, the back-face interior style attribute is used to render surfaces that are back-facing with respect to the viewing position.

The default back-face interior style is **PEXInteriorStyleHollow**.

If the distinguish attribute is **True**, the back-face interior style index attribute is used to choose a pattern or hatch style with which back-facing surfaces are rendered, if the back-face interior style is **PEXInteriorStylePattern** or **PEXInteriorStyleHatch**, respectively. This attribute is not used with other back-face interior styles.

The default back-face interior style index is one.

If the distinguish attribute is **True**, the back-face surface color attribute is used to render surfaces that are back-facing with respect to the viewing position.

The default back-facing surface color has color type **PEXColorTypeIndexed** and a color value of one.

If the distinguish attribute is **True**, the back-face surface reflection attributes are used instead of the reflection attributes to render surfaces that are back-facing with respect to the viewing position.

The default back-face surface reflection attributes are: ambient coefficient = 1.0, diffuse coefficient = 1.0, specular coefficient = 1.0, specular concentration = 0.0, transmission coefficient = 0.0, specular color type = **PEXColorTypeIndexed** and color value of one.

If the distinguish attribute is **True**, the back-face surface reflection model attribute is used instead of the surface reflection model attribute to render surfaces that are back-facing with respect to the viewing position.

The default back-face reflection model is **PEXReflectionNone**.

If the distinguish attribute is **True**, the back-face surface interpolation attribute is used instead of the surface interpolation attribute to render surfaces that are back-facing with respect to the viewing position.

The default back-face surface interpolation method is **PEXSurfaceInterpNone**.

The surface approximation attribute only affects non-uniform rational B-spline surfaces. Supported surface approximation methods are inquirable via **PEXGetEnumTypeInfo**. If the surface approximation method is **PEXApproxImpDep**, an implementation-dependent method is used to approximate surfaces in the u and v parameter dimensions. The tolerance values are not used for this method. For a method of **PEXApproxConstantBetweenKnots**, the surface is tessellated with equal parametric increments between successive pairs of knots. When the surface includes knots and the tolerance is negative, the surface is evaluated at just the parametric bounds, plus all knots which are between the parametric bounds. If the tolerance is positive, the surface is evaluated at the number of points between the knots, but only if these parametric values are between the parametric bounds. If the method is **PEXApproxWCChordalSize**, the surface is tessellated until the maximum length chord measured in world coordinates for both u and v parameter directions is less than the corresponding tolerance. If the method is **PEXApproxNPCChordalSize**, the surface is tessellated until the maximum length chord measured in normalized projection coordinates for both u and v parameter directions is less than the corresponding tolerance. If the method is **PEXApproxDCChordalSize**, the surface is tessellated until the maximum length chord measured in device coordinates for both u and v parameter directions is less than the corresponding tolerance. If the method is **PEXSurfaceApproxWCPlanarDev**, the surface is tessellated until the maximum deviation, in world coordinates, between the plane and the surface is less than the tolerance. If the method is **PEXSurfaceApproxNPCPlanarDev**, the surface is tessellated until the maximum deviation in normalized projection coordinates between the plane and the surface is less than the tolerance. If the method is **PEXSurfaceApproxDCPlanarDev**, the surface is tessellated until the maximum deviation in device coordinates between the plane and the surface is less than the tolerance. If the method is **PEXApproxWCRelative**, the u tolerance value must be between 0 and 1. The v tolerance value is not used for this method. The u tolerance specifies an approximation value which indicates a relative quality of rendering to be maintained independent of scaling in world coordinates. Values closer to 1 specify a better relative quality. If the method is **PEXApproxNPCRelative**, the u tolerance value must be between 0 and 1. The v tolerance value is not used for this method. The u tolerance specifies an approximation value which indicates a relative quality of rendering to be maintained independent of scaling in normalized projection coordinates. Values closer to 1 specify a better relative quality. If the method is **PEXApproxDCRelative**, the u tolerance value must be between 0 and 1. The v tolerance value is not used for this method. The u tolerance specifies an approximation value which indicates a relative quality of rendering to be maintained independent of scaling in device coordinates. Values closer to 1 specify a better relative quality.

The default surface approximation method is **PEXApproxImpDep**.

If the culling mode attribute is **PEXBackFaces**, all back-facing surfaces will be culled and only front-facing surfaces will be rendered. If the culling mode is **PEXFrontFaces**, all front-facing surfaces will be culled and only back-facing surfaces will be rendered. If the culling mode is **PEXNone**, both front- and back-facing fill areas will be rendered.

The default culling mode is **PEXNone**.

The facet distinguishing flag attribute selects whether back-facing surfaces are rendered with the back-face surface attributes or the front-face surface attributes. If distinguish is **True**, then back-face attributes are used to render the surface. If distinguish is **False**, then front-face attributes are used to render the surface.

The default face distinguish flag is **False**.

The pattern size attribute determines the scaling of the pattern used to render fill areas when the interior style or back-facing interior style attribute is **PEXInteriorStylePattern**. The value <width 0.0> will be used as the pattern width vector, and the value <0.0 height> will be used as the pattern height vector.

The default pattern size is <1.0, 1.0>.

The pattern reference point attribute determines the position, in model coordinates, of one corner of the pattern box used to render fill areas when the interior style or back-facing interior style attribute is **PEXInteriorStylePattern**.

The default pattern reference point is <0.0, 0.0, 0.0>.

The pattern reference vector 1 attribute determines the direction, in model coordinates, onto which a pattern's width will be mapped, when the interior style or the back-facing interior style attribute is **PEXInteriorStylePattern**.

The default pattern reference vector 1 is <1.0, 0.0, 0.0>.

The pattern reference vector 2 attribute determines the direction, in model coordinates, onto which a pattern's height will be mapped, when the interior style or the back-facing interior style attribute is **PEXInteriorStylePattern**.

The default pattern reference vector 2 is <0.0, 1.0, 0.0>.

The interior bundle index attribute contains the lookup table index used to obtain bundled surface attributes from the interior bundle table.

The default interior bundle index is one.

The parametric surface characteristics attribute controls the surface characteristics of non-uniform B-spline surfaces. The supported parametric surface characteristics are obtained by calling **PEXGetEnumTypeInfo**.

The default parametric surface characteristics type is **PEXPSCNone**.

4.5. Surface Edge Attributes

If the surface edges flag is **PEXOn**, surface edges are drawn. If it is **PEXOff**, surface edges are not drawn.

The default surface edges flag is **PEXOff**.

The surface edge type attribute specifies the style to use when drawing edges of surface primitives. Supported surface edge types are inquirable via **PEXGetEnumTypeInfo**. If the edge type is **PEXSurfaceEdgeSolid**, edges are drawn as solid, unbroken lines. If the edge type is **PEXSurfaceEdgeDashed**, edges are drawn using dashed lines. If the edge type is **PEXSurfaceEdgeDotted**, edges are drawn using dotted lines. If the edge type is **PEXSurfaceEdgeDashDot**, edges are drawn using alternating dots and dashes.

The default surface edge type is **PEXSurfaceEdgeSolid**.

The surface edge width attribute is the scale factor applied to the nominal width of the surface edges when they are rendered. Edge width is applied in 2D device coordinates after the edge has been transformed from 3D space to 2D device space. It is multiplied by the nominal edge width (see **PEXGetImpDepConstants**) to determine the actual width of the surface edge to be drawn.

The default surface edge width scale factor is 1.0.

The surface edge color attribute contains the color value used when drawing edges of surface primitives.

The default surface edge color has color type **PEXColorTypeIndexed** and a color value of one.

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

The default edge bundle index is one.

4.6. Geometry Transformation Attributes

The local transformation matrix attribute contains the local modeling transformation matrix used when drawing output primitives.

The default local transformation matrix is the identity matrix.

The global transformation matrix attribute contains the global modeling transformation matrix used when drawing output primitives. The current local modeling matrix and the current global modeling matrix are concatenated to form the composite modeling transformation matrix. The composite modeling transformation matrix is used to transform geometry in modeling coordinates into world coordinates.

The default global transformation matrix is the identity matrix.

The modeling clipping flag attribute determines whether output primitives are clipped against the modeling clipping volume.

The default model clipping flag is **PEXNoClip**.

The model clip volume attribute specifies the volume to which output primitives will be clipped, if the modeling clipping flag is **True**.

The default model clipping volume is an empty list (i.e. no half-spaces specified).

The view index attribute contains the lookup table index used to obtain viewing attributes from the view table. The view table contains the attribute that defines the view mapping transformation to be used (mapping), the attribute that defines the view orientation (projection) transformation to be used, and the attribute containing the min/max values of the clipping volume (clip limits). The view table also contains a mask (clip flags) indicating whether clipping should be performed at the front, back, and sides of the clipping volume.

The default view index is zero.

4.7. Color Transformation Attributes

The light source state attribute specifies which lights are enabled. The supported types of lights are obtained by calling **PEXGetEnumTypeInfo**. The maximum number of simultaneously active non-ambient lights are queried by calling **PEXGetImpDepConstants**. The size of the light table is obtained by calling **PEXGetTableInfo**.

The default light state is an empty list (all lights are disabled).

The depth-cue index attribute determines the depth-cue lookup table entry used to determine depth-cueing parameters.

The default depth-cue index is zero.

The color approximation index attribute determines the color approximation lookup table entry used for mapping rendering color to device color.

The default color approximation index is zero.

The rendering color model attribute controls which color model is used for lighting, shading and depth-cueing computations. Supported rendering color models are inquirable via **PEXGetEnumTypeInfo**.

The default rendering color model is **PEXRenderingColorModelImpDep**.

4.8. Aspect Source Flags

The bits in the aspect source flag (ASF) values bitmask contain an ASF value for each attribute obtained from a lookup (bundle) table. If the value is **PEXIndividual** during rendering, the value for the corresponding attribute is obtained directly from the current rendering pipeline state. If the value for the ASF is **PEXBundled**, the value for the attribute is obtained from the appropriate bundle lookup table. The default value for all ASF values is **PEXIndividual**.

A bitmask of ASF enables is also used when setting the ASF values. For each bit in the enables bitmask which is set to one, the corresponding value is taken from the ASF values bitmask and set in the pipeline context. The individual bits in both the ASF values and ASF enables bitmasks are referenced using the following constants:

PEXASFMarkerType
PEXASFMarkerScale
PEXASFMarkerColor
PEXASFTextFontIndex
PEXASFTextPrec
PEXASFCharExpansion
PEXASFCharSpacing

PEXASFTextColor
PEXASFLineType
PEXASFLineWidth
PEXASFLineColor
PEXASFCurveApprox
PEXASFPolylineInterp
PEXASFInteriorStyle
PEXASFInteriorStyleIndex
PEXASFSurfaceColor
PEXASFSurfaceInterp
PEXASFReflectionModel
PEXASFReflectionAttr
PEXASFBFInteriorStyle
PEXASFBFInteriorStyleIndex
PEXASFBFSurfaceColor
PEXASFBFSurfaceInterp
PEXASFBFReflectionModel
PEXASFBFReflectionAttr
PEXASFSurfaceApprox
PEXASFSurfaceEdges
PEXASFSurfaceEdgeType
PEXASFSurfaceEdgeWidth
PEXASFSurfaceEdgeColor

4.9. Miscellaneous Attributes

The pick identifier attribute contains the pick id used with picking operations. The pick identification is set to values which make it convenient for applications to identify the picked primitive(s).

The default pick identification is zero.

The HLHSR (hidden-line, hidden-surface removal) identifier attribute has no meaning for any standard HLHSR mode except **PEXHLHSRZBufferID**, where it enables or disables HLHSR. When the renderer's HLHSR mode is set to **PEXHLHSRZBufferID**, a value of **PEXHLHSRIDDisable** for the HLHSR identifier will disable z-buffering and a value of **PEXHLHSRIDEnable** will enable z-buffering. For non-standard HLHSR modes (returned by **PEXGetEnumTypeInfo**), the meaning of HLHSR identifier is implementation-dependent.

The default HLHSR identifier is zero. If an application wanted to have the renderer automatically z-buffer when the HLHSR mode was set to **PEXHLHSRZBufferID**, the application could change the pipeline context HLHSR identifier attribute to **PEXHLHSRIDEnable**. This would then alleviate the need for an **PEXSetHLHSRID** attribute call after every **PEXBeginRendering** when using immediate mode rendering.

The name set attribute contains a reference to a name set resource. The contents of the specified name set are used to initialize a rendering pipeline's name set attribute. The contents of the rendering pipeline's name set is then modified using the **PEXAddToNameSet** and **PEXRemoveFromNameSet** output commands.

The default name set is the empty set.

4.10. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* pipeline context attributes */
```

```
typedef struct {
    PEXEnumTypeIndex    marker_type;      /* PEXMarkerDot, PEXMarkerCross, */
                                 /* PEXMarkerAsterisk, PEXMarkerCircle, PEXMarkerX *
    float               marker_scale;
    PEXColorSpecifier   marker_color;
    PEXTableIndex       marker_bundle_index;
    PEXTableIndex       text_font;
    unsigned short      text_precision;       /* PEXStringPrecision, PEXCharPrecisio
    float               char_expansion;
    float               char_spacing;
    PEXColorSpecifier   text_color;
    float               char_height;
    PEXVector2D         char_up_vector;
    unsigned short      text_path;        /* PEXPathRight, PEXPathLeft, PEXPathUp, PE
    PEXTextAlignment    text_alignment;
    float               atext_height;
    PEXVector2D         atext_up_vector;
    unsigned short      atext_path;       /* PEXPathRight, PEXPathLeft, PEXPathUp, PE
    PEXTextAlignment    atext_alignment;
    PEXEnumTypeIndex    atext_style;      /* PEXATextNotConnected, PEXATextConnected
    PEXTableIndex       text_bundle_index;
    PEXEnumTypeIndex    line_type;            /* PEXLineTypeSolid, PEXLineTypeDashed
                                 /* PEXLineTypeDotted, PEXLineTypeDashDot */
    float               line_width;
    PEXColorSpecifier   line_color;
    PEXCurveApprox      curve_approx;
    PEXEnumTypeIndex    polyline_interp;      /* PEXPolylineInterpNone, PEXPolylineIn
    PEXTableIndex       line_bundle_index;
    PEXEnumTypeIndex    interior_style;       /* PEXInteriorStyleHollow, PEXInteriorS
                                 /* PEXInteriorStylePattern, PEXInteriorStyleHatch,
                                 /* PEXInteriorStyleEmpty */
    PEXTypeOrTableIndex interior_style_index;
    PEXColorSpecifier   surface_color;
    PEXReflectionAttributes   reflection_attr;
    PEXEnumTypeIndex    reflection_model; /* PEXReflectionNone, PEXReflectionAmbient,
                                 /* PEXReflectionDiffuse, PEXReflectionSpecular */
    PEXEnumTypeIndex    surface_interp;       /* PEXSurfaceInterpNone, PEXSurfaceInte
                                 /* PEXSurfaceInterpDotProduct, PEXSurfaceInterpNorm
    PEXEnumTypeIndex    bf_interior_style;    /* PEXInteriorStyleHollow, PEXInteriorS
                                 /* PEXInteriorStylePattern, PEXInteriorStyleHatch,
                                 /* PEXInteriorStyleEmpty */
    PEXTypeOrTableIndex bf_interior_style_index;
    PEXColorSpecifier   bf_surface_color;
    PEXReflectionAttributes   bf_reflection_attr;
    PEXEnumTypeIndex    bf_reflection_model;  /* PEXReflectionNone, PEXReflectionAmbi
                                 /* PEXReflectionDiffuse, PEXReflectionSpecular */
    PEXEnumTypeIndex    bf_surface_interp;    /* PEXSurfaceInterpNone, PEXSurfaceInte
                                 /* PEXSurfaceInterpDotProduct, PEXSurfaceInterpNorm
    PEXSurfaceApprox    surface_approx;
    unsigned short      culling_mode;     /* PEXNone, PEXBackFaces, PEXFrontFaces */
    Bool                distinguish;      /* True or False */
    PEXCoord2D          pattern_size;
    PEXCoord            pattern_ref_point;
    PEXVector           pattern_ref_vec1;
```

```
    PEXVector           pattern_ref_vec2;
    PEXTableIndex       interior_bundle_index;
    PEXSwitch           surface_edges;        /* True or False */
    PEXEnumTypeIndex    surface_edge_type;    /* PEXSurfaceEdgeSolid, PEXSurfaceEdge
                                 /* PEXSurfaceEdgeDotted, PEXSurfaceEdgeDashDot */
    float               surface_edge_width;
    PEXColorSpecifier   surface_edge_color;
    PEXTableIndex       edge_bundle_index;
    PEXMatrix           local_transform;
    PEXMatrix           global_transform;
    unsigned short      model_clip;       /* PEXClip, PEXNoClip */
    PEXModelClipVolume  model_clip_volume;
    PEXTableIndex       view_index;
    PEXListOfLight      light_state;
    PEXTableIndex       depth_cue_index;
    PEXBitmask          asf_enables;
    PEXBitmask          asf_values;
    long                pick_id;
    unsigned long       hlhsr_id;         /* PEXHLHSRIDDisable, PEXHLHSRIDEnable */
    PEXNameSet          name_set;
    PEXTableIndex       color_approx_index;
    PEXEnumTypeIndex    rendering_color_model;    /* PEXRenderingColorModelImpDep, *
                         /* PEXRenderingColorModelRGB, PEXRenderingColorModelCIE, */
                         /* PEXRenderingColorModelHSV, PEXRenderingColorModelHLS */
    PEXPSCSpecifier     para_surf_char;
} PEXPCAttributes;

typedef short           PEXEnumTypeIndex;
typedef unsigned short  PEXTableIndex;
typedef unsigned short  PEXTypeOrTableIndex;
typedef unsigned char   PEXSwitch;
typedef float           PEXMatrix[4][4];
typedef unsigned long   PEXBitmask;
typedef XID             PEXNameSet;

typedef struct {
    PEXColorType        type;        /* PEXColorTypeIndexed, PEXColorTypeRGB, */
                                 /* PEXColorTypeCIE, PEXColorTypeHSV, */
                                 /* PEXColorTypeHLS, PEXColorTypeRGB8, */
                                 /* PEXColorTypeRGB16 */
    unsigned short      reserved;
    PEXColor            value;
} PEXColorSpecifier;

typedef short           PEXColorType;

typedef union {
    PEXColorIndexed     indexed;
    PEXColorRGB         rgb;
    PEXColorHSV         hsv;
    PEXColorHLS         hls;
    PEXColorCIE         cie;
    PEXColorRGB8        rgb8;
    PEXColorRGB16       rgb16;
```

```
} PEXColor;

typedef struct {
    PEXTableIndex       index;
    unsigned short      reserved;
} PEXColorIndexed;

typedef struct {
    float               red;
    float               green;
    float               blue;
} PEXColorRGB;

typedef struct {
    float               hue;
    float               saturation;
    float               value;
} PEXColorHSV;

typedef struct {
    float               hue;
    float               lightness;
    float               saturation;
} PEXColorHLS;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXColorCIE;

typedef struct {
    unsigned char       red;
    unsigned char       green;
    unsigned char       blue;
    unsigned char       reserved;
} PEXColorRGB8;

typedef struct {
    unsigned short      red;
    unsigned short      green;
    unsigned short      blue;
    unsigned short      reserved;
} PEXColorRGB16;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXVector;

typedef struct {
    float               x;
    float               y;
```

```
} PEXVector2D;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    float               x;
    float               y;
} PEXCoord2D;

typedef struct {
    unsigned short      vertical;     /* PEXVAlignNormal, PEXVAlignTop, PEXVAlignCap,
                              /* PEXVAlignHalf, PEXVAlignBase, PEXVAlignBottom */
    unsigned short      horizontal;   /* PEXHAlignNormal, PEXHAlignLeft, */
                              /* PEXHAlignCenter, PEXHAlignRight */
} PEXTextAlignment;

typedef struct {
    PEXEnumTypeIndex    method;           /* See PEXGetEnumTypeInfo */
    unsigned short      reserved;
    float               tolerance;
} PEXCurveApprox;

typedef struct {
    float               ambient;
    float               diffuse;
    float               specular;
    float               specular_conc;
    float               transmission;
    PEXColorSpecifier   specular_color;
} PEXReflectionAttributes;

typedef struct {
    PEXEnumTypeIndex    method;           /* See PEXGetEnumTypeInfo */
    unsigned short      reserved;
    float               u_tolerance;
    float               v_tolerance;
} PEXSurfaceApprox;

typedef struct {
    unsigned short      count;                  /* number of half spaces */
    PEXHalfSpace        *half_spaces;
} PEXModelClipVolume;

typedef struct {
    PEXCoord            point;
    PEXVector           vector;
} PEXHalfSpace;

typedef struct {
    unsigned short      count;                  /* number of lights */
```

```
    PEXTableIndex       *indices;
} PEXListOfLight;

typedef struct {
    short               type;        /* PEXPSCNone, PEXPSCImpDep, PEXPSCIsoCurves, *
                                 /* PEXPSCMCLevelCurves, PEXPSCWCLevelCurves */
    PEXPSCData          psc;
} PEXPSCSpecifier;

typedef union {
    PEXPSCIsoparametricCurves iso_curves;
    PEXPSCLevelCurves       level_curves;
    PEXPSCImpDepData        imp_dep;
} PEXPSCData;

typedef struct {
    unsigned short      placement_type;
    unsigned short      reserved;
    unsigned short      u_count;
    unsigned short      v_count;
} PEXPSCIsoparametricCurves;

typedef struct {
    PEXCoord            origin;
    PEXVector           direction;
    unsigned short      count;                  /* number of parameters */
    unsigned short      reserved;
    float               *parameters;
} PEXPSCLevelCurves;

typedef struct {
    unsigned short      length;
    char                *data;
} PEXPSCImpDepData;
```

PEXChangePipelineContext - Change Pipeline Context

Synopsis

void PEXChangePipelineContext(Display **display*, PEXPipelineContext *context*, unsigned long **value_mask*, PEXPCAttributes **values*)

Arguments

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

Returns

None

Description

This function changes components of the specified pipeline context to the values specified. The value mask indicates which attribute values are specified. **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** can be called to setup the value mask.

Data Structures

```
typedef XID             PEXPipelineContext;
```

See also the *Common Data Structures* section.

Errors

BadPEXColorType	The specified color type is invalid or unsupported.
BadPEXNameSet	The specified name set resource identifier 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, **PEXGetPipelineContext**, **PEXSetPCAttributeMask**

PEXCopyPipelineContext - Copy Pipeline Context

Synopsis

void PEXCopyPipelineContext(Display **display*, unsigned long **value_mask*, PEXPipelineContext *src_context*, PEXPipelineContext *dst_context*)

Arguments

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

value_mask A pointer to an array of three unsigned long.

src_context The resource identifier of the source pipeline.

dst_context The resource identifier of the destination pipeline context.

Returns

None

Description

This function copies attributes from the source pipeline context to the destination pipeline context. Both must already exist as valid pipeline context resources. Only the attributes indicated by the value mask are copied and the remainder of the attributes are left as they were. **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** can be called to setup the value mask.

Data Structures

```
typedef XID             PEXPipelineContext;
```

Errors

BadPEXPipelineContext
A specified pipeline context resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreatePipelineContext, **PEXSetPCAttributeMask**

PEXCreatePipelineContext - Create Pipeline Context

Synopsis

PEXPipelineContext PEXCreatePipelineContext(Display **display*, unsigned long **value_mask*, PEXPCAttributes **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
value_mask	A pointer to an array of three unsigned long.
values	A pointer to values to override default attribute values in the new pipeline context.

Returns

The resource identifier of the newly-created pipeline context.

Description

This function creates a pipeline context and returns its resource identifier. The value mask indicates the values specified. Attribute values not specified will be initialized to the default values. **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** can be called to setup the value mask.

Data Structures

```
typedef XID             PEXPipelineContext;
```

See also the *Common Data Structures* section.

Errors

BadAlloc	The server failed to allocate the resource.
BadPEXColorType	The specified color type is invalid or unsupported.
BadPEXNameSet	A specified name set resource identifier is invalid.
BadValue	A specified value is out of range, or an invalid bit is set in the value mask.

See Also

PEXFreePipelineContext

PEXFreePCAttributes - Free Storage Returned by PEXGetPipelineContext

Synopsis

void PEXFreePCAttributes(PEXPCAttributes **values*)

Arguments

values A pointer to the pipeline context attribute values.

Returns

None

Description

This function deallocates memory returned by **PEXGetPipelineContext**.

Errors

None

See Also

PEXGetPipelineContext

PEXFreePipelineContext - Free Pipeline Context

Synopsis

void PEXFreePipelineContext(Display **display*, PEXPipelineContext *context*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
context	The resource identifier of the pipeline context.

Returns

None

Description

This function deletes the association between the pipeline context resource identifier and the pipeline context. The pipeline context is freed when no other resource references it.

Errors

BadPEXPipelineContext
The specified pipeline context resource identifier is invalid.

See Also

PEXCreatePipelineContext

PEXGetPipelineContext - Get Pipeline Context Attributes

Synopsis

PEXPCAttributes *PEXGetPipelineContext(Display **display*, PEXPipelineContext *context*, unsigned long **value_mask*)

Arguments

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

context The resource identifier of the pipeline context.

value_mask A pointer to an array of three unsigned long.

Returns

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

Description

This function returns the requested attribute values of the pipeline context. The value mask indicates which attribute values are requested. **PEXSetPCAttributeMask** and **PEXSetPCAttributeMaskAll** can be called to setup the value mask. PEXlib allocates the memory for the return value. **PEXFreePCAttributes** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXPipelineContext;
```

See also the *Common Data Structures* section.

Errors

BadPEXPipelineContext
The specified pipeline context resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreatePipelineContext, **PEXChangePipelineContext**

PEXSetPCAttributeMask - Macro to Setup Pipeline Context Attributes Value Mask

Synopsis

PEXSetPCAttributeMask(*mask*, *attr*)

Arguments

mask	The address of the value mask - an array of three unsigned long.
attr	A single pipeline context attribute bitmask constant.

Description

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

The following attribute bitmask constants must be used:

PEXPCMarkerType
PEXPCMarkerScale
PEXPCMarkerColor
PEXPCMarkerBundleIndex
PEXPCTextFont
PEXPCTextPrecision
PEXPCCharExpansion
PEXPCCharSpacing
PEXPCTextColor
PEXPCCharHeight
PEXPCCharUpVector
PEXPCTextPath
PEXPCTextAlignment
PEXPCATextHeight
PEXPCATextUpVector
PEXPCATextPath
PEXPCATextAlignment
PEXPCATextStyle
PEXPCTextBundleIndex
PEXPCLineType
PEXPCLineWidth
PEXPCLineColor
PEXPCCurveApprox
PEXPCPolylineInterp
PEXPCLineBundleIndex
PEXPCInteriorStyle
PEXPCInteriorStyleIndex
PEXPCSurfaceColor
PEXPCReflectionAttr
PEXPCReflectionModel
PEXPCSurfaceInterp
PEXPCBFInteriorStyle
PEXPCBFInteriorStyleIndex
PEXPCBFSurfaceColor
PEXPCBFReflectionAttr
PEXPCBFReflectionModel
PEXPCBFSurfaceInterp
PEXPCSurfaceApprox
PEXPCCullingMode
PEXPCDistinguishFlag

PEXPCPatternSize
PEXPCPatternRefPoint
PEXPCPatternRefVec1
PEXPCPatternRefVec2
PEXPCInteriorBundleIndex
PEXPCSurfaceEdgeFlag
PEXPCSurfaceEdgeType
PEXPCSurfaceEdgeWidth
PEXPCSurfaceEdgeColor
PEXPCEdgeBundleIndex
PEXPCLocalTransform
PEXPCGlobalTransform
PEXPCModelClip
PEXPCModelClipVolume
PEXPCViewIndex
PEXPCLightState
PEXPCDepthCueIndex
PEXPCASFValues
PEXPCPickID
PEXPCHLHSRIdentifier
PEXPCNameSet
PEXPCColorApproxIndex
PEXPCRenderingColorModel
PEXPCParaSurfCharacteristics

Note that this macro does multiple evaluations of the value for attr.

Errors

None

See Also

PEXCreatePipelineContext, **PEXChangePipelineContext**, **PEXCopyPipelineContext**, **PEXGetPipelineContext**

PEXSetPCAttributeMaskAll - Macro to Set All Pipeline Context Attributes in Value Mask

Synopsis

PEXSetPCAttributeMaskAll(*mask*)

Arguments

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 pipeline context attributes. This macro will set all valid bits in the mask.

Errors

None

See Also

PEXCreatePipelineContext, **PEXChangePipelineContext**, **PEXCopyPipelineContext**, **PEXGetPipelineContext**

Chapter 5

Renderers

PEX deals with three-dimensional (geometric) objects and color values specified by clients. Rendering is the process that converts geometric information into raster (2D) coordinates and converts color values into colors or intensities the device can display. Geometric or color data is rendered to an X drawable after it has been transformed into raster data. The rendering pipeline is an abstraction used to describe the order and the semantics of functions occurring during the rendering process.

A renderer is a PEX resource containing the functionality necessary to perform rendering. A renderer also contains attributes, some of which are resource identifiers for various tables and name sets, and for the pipeline context from which initial rendering pipeline attributes are copied. When a renderer is created, if lookup table resources are not specified for some lookup tables, the default value will be no lookup table. When rendering, if a lookup table resource doesn't exist in the renderer, the default values will be used. These defaults are given in the corresponding description of the output commands in *Chapter 2: Output Commands*. The default name set is an empty set. If no pipeline context is specified, the default pipeline context values will be used. These defaults are given in *Chapter 4: Pipeline Contexts*.

If a pipeline context, lookup table or name set resource is bound to the renderer and then freed, inquiries on the renderer attributes will return the value of **PEXAlreadyFreed** for those particular resource identifiers.

Output commands sent to a renderer are rendered immediately. Output commands can also be stored in a structure resource, and then the structure traversed by a renderer. In this way, renderers support both concepts of immediate-mode and retained- or structure-mode graphics.

PEXBeginRendering is called to prepare a renderer for immediate-mode rendering. (**PEXRenderNetwork** also implicitly performs a "begin rendering".) The PEX server implementation may allow certain renderer attributes to be bound only at the time of **PEXBeginRendering** or at any time during the lifetime of the renderer resource. **PEXGetRendererDynamics** can be called to obtain information about how renderer attributes are bound for a particular PEX implementation.

If a drawable associated with a renderer is destroyed while a renderer is rendering, an implicit **PEXEndRendering** is performed by the server with the flush flag equal to **False** in order to return the renderer to the **PEXIdle** renderer state. All subsequent output and traversal requests to the renderer are ignored and no error is generated.

If a drawable associated with a renderer is resized while a renderer is rendering, an implicit **PEXEndRendering** is performed by the server with the flush flag equal to **False** in order to return the renderer to the **PEXIdle** renderer state. All subsequent output and traversal requests to the renderer are ignored until the next request to the renderer to begin rendering or begin a pick operation and no error is generated.

If a drawable that is associated with a renderer is moved, exposed or occluded while the renderer is rendering, the server will continue to process output commands and requests using the new drawable attributes until the rendering operation is explicitly or implicitly terminated.

A renderer also manages the HLHSR buffer needed for some HLHSR (hidden-line/hidden-surface removal) algorithms. When the renderer begins rendering (either an explicit or implicit **PEXBeginRendering**) and the HLHSR mode indicates z-buffering should be performed, a z-buffer suitable for use with the specified drawable is allocated, if necessary, and bound to the renderer. A new z-buffer is allocated only if one does not already exist for the renderer, or if the z-buffer currently bound to the renderer is not suitable for the specified drawable. If the z-buffer could not be allocated, an allocation error is generated and the rendering is aborted. If the clear z flag is enabled, or if a new z-buffer is allocated, then the z-buffer will be cleared to infinity in the region specified by the renderer's current clip list. The value of infinity is implementation-dependent. The z-buffer is not deallocated when an explicit or implicit **PEXEndRendering** occurs. The z-buffer remains bound to the renderer until the renderer is freed or until a new one is allocated as described above. This means that when the next **PEXBeginRendering** occurs, the z-buffer will be re-used if possible. This allows the application to control operations that require the z-buffer to remain intact between rendering traversals and immediate-mode primitives. Some reasons why the z-buffer may not be re-usable are the specification of a different drawable or the previous drawable was resized.

Also, the renderer resource does not contain a double-buffer mode, whereas the PHIGS workstation resource does. This does not preclude the use of the PEX renderer resource in a double-buffered drawable. It is intended that applications will use the *Multi-Buffer Extension* to X (MBX) for double-buffering with a PEX renderer resource.

The next section describes the attributes of a renderer and their effects.

5.1. Renderer Attributes

Each renderer has the following attributes that are not directly modifiable (via **PEXChangeRenderer**) but are inquirable (via **PEXGetRendererAttributes**):

renderer state
current path

The renderer state attribute is used to identify whether a renderer is currently **PEXRendering**, **PEXPicking**, or **PEXIdle**. Calling **PEXBeginRendering** puts a renderer in the **PEXRendering** state. See *Chapter 6: Renderer Picking* for more information about the picking renderer state. Output primitives or attributes are processed by a renderer only when the renderer state is **PEXRendering** or **PEXPicking**. Calling **PEXEndRendering** puts a renderer in the **PEXIdle** state. Output primitives sent to an idle renderer are ignored.

The current path attribute contains a list of element references which indicate paths for use by an application doing immediate-mode rendering, or traversing structures via the "execute structure" output command. Applications can use **PEXBeginStructure** and **PEXEndStructure** to maintain the current path during immediate-mode rendering. The current path attribute is initialized to structure id 0 and offset 0.

Each renderer has the following attributes that are modifiable (via **PEXChangeRenderer**) and inquirable (via **PEXGetRendererAttributes**):

pipeline context
color approximation table
color table
depth cue table
edge bundle
interior bundle
light table
line bundle
marker bundle
pattern table
text bundle
text font table
view table
highlight inclusion
highlight exclusion
invisibility inclusion
invisibility exclusion
pick inclusion
pick exclusion
HLHSR mode
NPC subvolume
viewport

clip list
pick start path
background color
clear image flag
clear z flag
echo mode

Each renderer has a set of state information (the renderer's pipeline state) that defines the behavior of the rendering pipeline. A renderer's pipeline state contains all attributes stored in a PEX pipeline context. **PEXBeginRendering** copies into the renderer's pipeline state the attributes in the pipeline context specified by the renderer's pipeline context attribute. If the pipeline context attribute does not contain the resource id of a valid pipeline context resource, the default values for pipeline context attributes are copied instead.

Each time a renderer encounters an "execute structure" command, the renderer's pipeline state is saved. When the called structure has been executed, this saved pipeline state is restored. Note that only the pipeline state values are saved and restored. The renderer state and current path attributes are not saved and restored, since they are not modifiable from within a structure.

A renderer's pipeline state attributes are modified by output attribute functions.

Each of the bundle and table attributes contains the resource id of the specified type of lookup table. If not set when the renderer is created, they are all NULL. If they are NULL during the rendering operation, the default attributes for the lookup table are used. Pipeline attributes whose ASFs are set to **PEXBundled** obtain their values from the edge bundle, interior bundle, line bundle, marker bundle and text bundle tables.

The view table provides clipping information and the orientation and mapping matrices. Indexed colors passed to the renderer are dereferenced using the color table. The depth-cue table provides depth-cueing information. The light table stores definitions for all light sources available to the renderer. The color approximation table contains the rules and hints for converting rendering pipeline colors into pixel values consistent with the visual and color map for the drawable. The text font table contains a list of X and PEX font resource identifiers. See *Chapter 3: Lookup Tables* for more details on the different types of lookup tables.

Each of the inclusion and exclusion attributes contains the resource id of a name set resource which is associated with each renderer as filters. If not set when the renderer is created, these filters are NULL, which means they contain no names.

Highlighting is the process of drawing attention to an output primitive by altering its visual attributes in an implementation-dependent way. Primitives treated as invisible are simply not sent through the rendering pipeline. Each filter consists of two name sets: an inclusion set and an exclusion set. Another name set is maintained as part of the renderer's pipeline state. Some output commands add and remove information from the pipeline's name set.

For a primitive to be highlighted, the pipeline's name set must contain a name (a 32-bit identifier) that also exists in the highlight inclusion name set. The highlighted primitive pipeline's name set must not contain names found in the highlight exclusion name set. For a primitive to be treated as invisible, the pipeline's name set must contain a name that also exists in the invisibility inclusion name set. The invisible primitive pipeline's name set must not contain names also found in the invisibility exclusion name set. In other words, to meet the filtering criteria for either highlighting or invisibility, the intersection of the name set with the inclusion set must not be empty, and the intersection of the name set with the exclusion set must be the empty set.

See *Chapter 6: Renderer Picking* for information on the pick filters.

The default HLHSR mode is **PEXHLHSROff.** Each renderer maintains an HLHSR mode indicating the hidden line and hidden surface algorithm used. Each renderer also maintains an HLHSR buffer used in the hidden line and hidden surface computations. When **PEXBeginRendering** is called, the HLHSR buffer is initialized based on the current value of the HLHSR mode. When **PEXEndRendering** is called, if the flush flag is **True**, unprocessed output primitives are rendered using the chosen HLHSR algorithm. If the flush flag is **False**, unprocessed output primitives are discarded.

The NPC subvolume attribute contains the normalized projection coordinates for the sub-volume space that is mapped to the viewport. The default NPC subvolume is [0.0, 1.0] in all three dimensions, x, y, z.

The viewport attribute describes the area within the drawable in which rendering may appear. The viewport coordinates are specified in device coordinates, which have an x,y offset (in pixels) from the lower-left corner of the drawable. Note that the y-component is the reverse of the X window system coordinate system. It is permissible to specify a viewport with boundaries outside of the drawable. Geometry that is mapped to values outside the drawable is clipped. Viewport z values must be in the range of 0.0 to 1.0, where z=0.0 maps to the device-coordinate z value representing objects that are furthest from the viewing position, and z=1.0 maps to the device-coordinate value representing objects that are closest to the viewing position. Depending on the dynamics of the viewport attribute (see **PEXGetRendererDynamics**), the values that define a viewport may be bound at any time, or they may take effect only at the time of **PEXBeginRendering**. Whenever the viewport values are bound, the viewport's use drawable flag is examined first. If it is set to **True**, the viewport width and height are obtained from the drawable's current width and height and the viewport z values are obtained from the values specified in the viewport structure. If the use drawable flag is **False**, the viewport width, height and z values are all set to the values in the viewport structure. The viewport is then set to the largest rectangle, anchored at the lower-left corner of the drawable, that achieves an isotropic mapping to the renderer's NPC subvolume. The viewport will remain in the same position relative to the lower-left hand corner of the drawable after a resize event has occurred. The default viewport size is implementation-dependent, and the default use drawable flag is **True**.

The clip list contains a list of rectangles in device coordinates that define the portions of the drawable in which rendering is allowed. If the list is empty, the renderer will render to the entire drawable area. If the list is not empty, only the area within the rectangles will be affected during rendering. The rectangles must be non-overlapping or the result is undefined and implementation-dependent. The rectangles in the clip list are defined in device coordinates. (0,0) is the lower left corner of the drawable and the xmax and ymax values are inclusive. These rectangles will remain in the same position relative to the lower left corner of the drawable after a resize has occurred. Note, if a z-buffering algorithm is used for HLHSR, only those z-values associated with pixels that are within the defined area of the clip list will be modified.

The pick start path is used as the starting path for "pick all". See *Chapter 6: Renderer Picking* for more information.

The background color is used to clear the drawable area when an explicit or implicit **PEXBeginRendering** occurs. The drawable will be cleared only if the clear image flag is enabled. If the clear image flag is disabled, the drawable will not be cleared. The default value for the clear image flag is **False**. The clip list will be taken into account when clearing the drawable. The initial index to the color approximation table is used to map the color to a device pixel value.

The clear z flag indicates whether the z-buffer should be cleared to infinity at the start of rendering. The value of infinity is implementation-dependent. The default value for the clear image flag is **True**. The clip list will be taken into account when clearing the z-buffer. This attribute is ignored if the HLHSR mode is not one that requires a z-buffer.

The echo mode indicates whether rendered primitives should be echoed, unechoed or rendered normally. The method used for echoing and unechoing primitives is implementation-dependent and there is no guarantee that the method will preserve temporal priority. Echoing takes precedence over highlighting. There is a standard PEX escape for setting an echo color that will provide more control over the echo method used, if the escape is supported. (See **PEXSetEchoColor**.)

5.2. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* renderer attributes */

typedef struct {
    PEXPipelineContext  pipeline_context;
    PEXStructurePath    current_path;
    PEXLookupTable      marker_bundle;
    PEXLookupTable      text_bundle;
    PEXLookupTable      line_bundle;
    PEXLookupTable      interior_bundle;
    PEXLookupTable      edge_bundle;
    PEXLookupTable      view_table;
    PEXLookupTable      color_table;
    PEXLookupTable      depth_cue_table;
    PEXLookupTable      light_table;
    PEXLookupTable      color_approx_table;
    PEXLookupTable      pattern_table;
    PEXLookupTable      text_font_table;
    PEXNameSet          highlight_incl;
    PEXNameSet          highlight_excl;
    PEXNameSet          invisibility_incl;
    PEXNameSet          invisibility_excl;
    int                 renderer_state;
    PEXEnumTypeIndex    hlhsr_mode;      /* PEXHLHSROff, PEXHLHSRZBuffer, */
                                 /* PEXHLHSRPainters, PEXHLHSRScanline, */
                                 /* PEXHLHSRHiddenLineOnly, */
                                 /* PEXHLHSRZBufferID */
    PEXNPCSubVolume     npc_subvolume;
    PEXViewport         viewport;
    PEXListOfClipRect   clip_list;
    PEXNameSet          pick_incl;
    PEXNameSet          pick_excl;
    PEXStructurePath    pick_start_path;
    PEXColorSpecifier   background_color;
    Bool                clear_image;     /* True or False */
    Bool                clear_z;         /* True or False */
    int                 echo_mode;           /* PEXNoEcho, PEXEcho, PEXUnEcho */
} PEXRendererAttributes;

typedef XID             PEXPipelineContext;
typedef XID             PEXLookupTable;
typedef XID             PEXNameSet;
typedef short           PEXEnumTypeIndex;

typedef struct {
    unsigned long       count;                /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;

typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;
```

```
typedef XID             PEXStructure;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord      min;
    PEXDeviceCoord      max;
    PEXSwitch           use_drawable;     /* True or False */
    unsigned char       reserved[3];
} PEXViewport;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef unsigned char   PEXSwitch;

typedef struct {
    unsigned short      count;          /* number of device rectangles */
    PEXDeviceRect       *rectangles;
} PEXListOfClipRect;

typedef struct {
    short               xmin;
    short               ymin;
    short               xmax;
    short               ymax;
} PEXDeviceRect;

typedef struct {
    PEXColorType        type;     /* PEXColorTypeIndexed, PEXColorTypeRGB, */
                             /* PEXColorTypeCIE, PEXColorTypeHSV, */
                             /* PEXColorTypeHLS, PEXColorTypeRGB8, */
                                  /* PEXColorTypeRGB16 */
    unsigned short      reserved;
    PEXColor            value;
} PEXColorSpecifier;

typedef short           PEXColorType;

typedef union {
    PEXColorIndexed     indexed;
    PEXColorRGB         rgb;
```

```
    PEXColorHSV         hsv;
    PEXColorHLS         hls;
    PEXColorCIE         cie;
    PEXColorRGB8        rgb8;
    PEXColorRGB16       rgb16;
} PEXColor;

typedef struct {
    PEXTableIndex       index;
    unsigned short      reserved;
} PEXColorIndexed;

typedef struct {
    float               red;
    float               green;
    float               blue;
} PEXColorRGB;

typedef struct {
    float               hue;
    float               saturation;
    float               value;
} PEXColorHSV;

typedef struct {
    float               hue;
    float               lightness;
    float               saturation;
} PEXColorHLS;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXColorCIE;

typedef struct {
    unsigned char       red;
    unsigned char       green;
    unsigned char       blue;
    unsigned char       reserved;
} PEXColorRGB8;

typedef struct {
    unsigned short      red;
    unsigned short      green;
    unsigned short      blue;
    unsigned short      reserved;
} PEXColorRGB16;
```

PEXAccumulateState - Accumulate Rendering Pipeline State

Synopsis

void PEXAccumulateState(Display **display*, PEXRenderer *renderer*, unsigned long *count*, PEXElementRef **elements*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of a renderer resource.
count	The number of elements.
elements	A pointer to the structure element reference path.

Returns

None

Description

This function accumulates the state that would be in effect if a traversal were done to the element specified in the path. If the renderer state is **PEXIdle**, the request is ignored.

The accumulation of rendering pipeline state begins with the current pipeline attributes. A linear traversal down the specified path is then made and the structure elements that lie along the specified path are examined in order. Any element that contains an output command that would modify the pipeline state (i.e. output attributes) is sent to the renderer for processing. All other output commands (i.e. output primitives and structure output commands) are skipped. The traversal is flat meaning that the current pipeline attributes will not be saved when a structure in the path is executed. However, the current path offset is incremented for each output command that is encountered during the state accumulation.

Data Structures

```
typedef XID             PEXRenderer;
typedef XID             PEXStructure;

typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;
```

Errors

BadPEXPath	The specified path is invalid.
BadPEXRenderer	The specified renderer resource identifier is invalid.

See Also

PEXBeginRendering, **PEXCreateStructure**, **PEXCreateRenderer**

PEXBeginRendering - Begin Rendering

Synopsis

void PEXBeginRendering(Display **display*, Drawable *drawable*, PEXRenderer *renderer*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
renderer	The resource identifier of a renderer.

Returns

None

Description

This function initializes the specified renderer and binds the specified drawable to it. Subsequent output primitive commands sent to renderer produce output to the specified drawable. The renderer's pipeline state is initialized to the values in its pipeline context, or to default values if there is no pipeline context. This function causes the renderer's renderer state to be set to **PEXRendering** and its current path to be set to <0,0>. The renderer's HLHSR mode is used to initialize hidden surface computations. (For example, the z-buffer is initialized if the HLHSR mode is set to **PEXHLHSRZBuffer** or **PEXHLHSRZBufferID**.)

If the renderer's state is **PEXRendering** or **PEXPicking** when this request is received, an implicit "end rendering or picking" request is performed before the "begin rendering" request is executed. Output commands received by a renderer are ignored if the state is not **PEXRendering** or **PEXPicking**.

Data Structures

```
typedef XID              PEXRenderer;
```

Errors

BadAlloc	The server failed to allocate the resource.
BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported, or the specified renderer resource was not created with a compatible drawable.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The specified renderer was in an invalid state.

See Also

PEXCreateRenderer, **PEXEndRendering**

PEXBeginStructure - Save Rendering Pipeline State

Synopsis

void PEXBeginStructure(Display **display*, PEXRenderer *renderer*, long *structure_id*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of a renderer.
structure_id	A value to be used as an application-specified structure identifier.

Returns

None

Description

Application programs can use this function to simulate the effects of the execute structure output command. This request saves the rendering pipeline attributes in the specified renderer. This request also increments the element offset of the last entry in the renderer's current path. The function then adds the structure id and an element offset of zero to the renderer's current path.

The rendering pipeline's global transform attribute is set to the matrix computed by concatenating the current local transform and current global transform matrices. The local transform matrix is then set to the identity matrix.

Data Structures

```
typedef XID             PEXRenderer;
```

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

See Also

PEXEndStructure, **PEXExecuteStructure**, **PEXCreateRenderer**

PEXChangeRenderer - Change Renderer Attributes

Synopsis

void PEXChangeRenderer(Display **display*, PEXRenderer *renderer*, unsigned long *value_mask*, PEXRendererAttributes **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer.
value_mask	A mask indicating the renderer attributes to be changed.
values	A pointer to new values for the renderer attributes.

Returns

None

Description

This function modifies attributes of the specified renderer resource. The value mask indicates the attribute values to be modified. The value mask is constructed by or'ing together the following constants:

PEXRAPipelineContext
PEXRAMarkerBundle
PEXRATextBundle
PEXRALineBundle
PEXRAInteriorBundle
PEXRAEdgeBundle
PEXRAViewTable
PEXRAColorTable
PEXRADepthCueTable
PEXRALightTable
PEXRAColorApproxTable
PEXRAPatternTable
PEXRATextFontTable
PEXRAHighlightIncl
PEXRAHighlightExcl
PEXRAInvisibilityIncl
PEXRAInvisibilityExcl
PEXRAHLHSRMode
PEXRANPCSubVolume
PEXRAViewport
PEXRAClipList
PEXRAPickIncl
PEXRAPickExcl
PEXRAPickStartPath
PEXRABackgroundColor
PEXRAClearImage
PEXRAClearZ
PEXRAEchoMode

Note that the renderer's current path and renderer state attributes cannot be set explicitly using this function.

Data Structures

```
typedef XID              PEXRenderer;
```

See also the *Common Data Structures* section.

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 identifier 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, **PEXGetRendererAttributes**, **PEXGetRendererDynamics**

PEXCreateRenderer - Create Renderer

Synopsis

PEXRenderer PEXCreateRenderer(Display **display*, Drawable *drawable*, unsigned long *value_mask*, PEXRendererAttributes **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
value_mask	A mask indicating the attribute values specified.
values	A pointer to values used to override default values in the new renderer resource.

Returns

The resource identifier of the newly-created renderer resource.

Description

This function creates a renderer and returns the resource identifier of the newly-created renderer resource. The value mask indicates which attribute values are specified. The value mask is constructed by or'ing together the following constants:

PEXRAPipelineContext
PEXRAMarkerBundle
PEXRATextBundle
PEXRALineBundle
PEXRAInteriorBundle
PEXRAEdgeBundle
PEXRAViewTable
PEXRAColorTable
PEXRADepthCueTable
PEXRALightTable
PEXRAColorApproxTable
PEXRAPatternTable
PEXRATextFontTable
PEXRAHighlightIncl
PEXRAHighlightExcl
PEXRAInvisibilityIncl
PEXRAInvisibilityExcl
PEXRAHLHSRMode
PEXRANPCSubVolume
PEXRAViewport
PEXRAClipList
PEXRAPickIncl
PEXRAPickExcl
PEXRAPickStartPath
PEXRABackgroundColor
PEXRAClearImage
PEXRAClearZ
PEXRAEchoMode

Fields not explicitly set have default values (see the beginning of the chapter). The renderer resource may only be used in conjunction with drawables that have the same root and depth as specified drawable.

Note that the renderer's current path and renderer state attributes cannot be set explicitly using this function.

Data Structures

```
typedef XID             PEXRenderer;
```

See also the *Common Data Structures* section.

Errors

BadAlloc The server failed to allocate the resource.

BadDrawable The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported, or the specified lookup table resource was not created with a drawable compatible with the specified drawable.

BadPEXLookupTable The specified lookup table resource identifier is invalid.

BadPEXNameSet The specified name set resource identifier 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

PEXFreeRenderer

PEXEndRendering - End Rendering

Synopsis

void PEXEndRendering(Display **display*, PEXRenderer *renderer*, int *flush*)

Arguments

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

renderer The resource identifier of a renderer.

flush **True** or **False** - specifying whether any pending output for renderer is to be rendered onto its associated drawable.

Returns

None

Description

This function terminates rendering on the specified renderer resource. If flush is **True**, pending output is rendered onto its associated drawable. If flush is **False**, pending output is discarded. In either case, the renderer state is set to **PEXIdle**. If the renderer state is **PEXIdle**, the request is ignored. If the renderer state is **PEXPicking**, then an error is generated.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXRendererState
The specified renderer was in an invalid state.

See Also

PEXCreateRenderer, **PEXBeginRendering**

PEXEndStructure - Restore Rendering Pipeline State

Synopsis

void PEXEndStructure(Display **display*, PEXRenderer *renderer*)

Arguments

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

renderer The resource identifier of the renderer.

Returns

None

Description

Application programs can use this function to simulate the side effects of the return from an "execute structure" output command. This request restores the last-saved rendering pipeline attributes in renderer. This request also removes the last element reference in the renderer's current path. Subsequent output commands cause the element offset of the element reference at the end of the list to be incremented.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXRendererState
There is no matching begin structure.

See Also

PEXBeginStructure, **PEXExecuteStructure**, **PEXCreateRenderer**

PEXFreeRenderer - Free Renderer

Synopsis

void PEXFreeRenderer(Display **display*, PEXRenderer *renderer*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer.

Returns

None

Description

This function deletes the renderer resource specified by renderer and frees the memory associated with it.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

See Also

PEXCreateRenderer

PEXFreeRendererAttributes - Free Storage Returned by PEXGetRendererAttributes

Synopsis

void PEXFreeRendererAttributes(PEXRendererAttributes **values*)

Arguments

values	A pointer to the renderer attribute values.

Returns

None

Description

This function deallocates memory returned by **PEXGetRendererAttributes**.

Errors

None

See Also

PEXGetRendererAttributes

PEXGetRendererAttributes - Get Renderer Attribute Values

Synopsis

PEXRendererAttributes *PEXGetRendererAttributes(Display **display*, PEXRenderer *renderer*, unsigned long *value_mask*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer to be queried.
value_mask	A mask indicating attributes to be returned from the renderer.

Returns

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

Description

This function returns attribute values from the specified renderer resource. The value mask indicates the attribute values to be returned. The value mask is constructed by or'ing together the following constants:

PEXRAPipelineContext
PEXRACurrentPath
PEXRAMarkerBundle
PEXRATextBundle
PEXRALineBundle
PEXRAInteriorBundle
PEXRAEdgeBundle
PEXRAViewTable
PEXRAColorTable
PEXRADepthCueTable
PEXRALightTable
PEXRAColorApproxTable
PEXRAPatternTable
PEXRATextFontTable
PEXRAHighlightIncl
PEXRAHighlightExcl
PEXRAInvisibilityIncl
PEXRAInvisibilityExcl
PEXRARendererState
PEXRAHLHSRMode
PEXRANPCSubVolume
PEXRAViewport
PEXRAClipList
PEXRAPickIncl
PEXRAPickExcl
PEXRAPickStartPath
PEXRABackgroundColor
PEXRAClearImage
PEXRAClearZ
PEXRAEchoMode

PEXlib allocates the memory for the returned renderer attribute values. **PEXFreeRendererAttributes** should be called to deallocate the memory.

Data Structures

```
typedef XID              PEXRenderer;
```

See also the *Common Data Structures* section.

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreateRenderer, **PEXChangeRenderer**, **PEXGetRendererDynamics**, **PEXFreeRendererAttributes**

PEXGetRendererDynamics - Get Renderer Attribute Modification Dynamics

Synopsis

Status PEXGetRendererDynamics(Display **display*, PEXRenderer *renderer*, unsigned long **tables_return*, unsigned long **name_sets_return*, unsigned long **attributes_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer.
tables_return	Returns a mask describing dynamics of lookup tables associated with the renderer.
name_sets_return	Returns a mask describing dynamics of name sets associated with the renderer.
attributes_return	Returns a mask describing dynamics of other attributes associated with the renderer.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns the modification dynamics for all of the attributes of the specified renderer resource. Each bit, in the returned bitmasks, indicates the dynamics for a particular renderer attribute. For each bit, a value of **PEXDynamic** indicates that the specified attribute may be modified at any time, and the change will take place immediately. A value of **PEXNotDynamic** indicates that the specified attribute may not be modified dynamically. In this case, the change is "pending" and will take effect at the next explicit or implicit **PEXBeginRendering**.

The pipeline context, pick start path, background color, clear image and clear z renderer attributes all take effect only at the time of "begin rendering or picking", and so do not have dynamics associated with them.

The bits in the tables bitmask are accessed with the following constants:

PEXRDTMarkerBundle
PEXRDTTextBundle
PEXRDTLineBundle
PEXRDTInteriorBundle
PEXRDTEdgeBundle
PEXRDTViewTable
PEXRDTColorTable
PEXRDTDepthCueTable
PEXRDTLightTable
PEXRDTColorApproxTable
PEXRDTPatternTable
PEXRDTTextFontTable
PEXRDTMarkerBundleContents
PEXRDTTextBundleContents
PEXRDTLineBundleContents
PEXRDTInteriorBundleContents
PEXRDTEdgeBundleContents
PEXRDTViewTableContents
PEXRDTColorTableContents
PEXRDTDepthCueTableContents
PEXRDTLightTableContents
PEXRDTColorApproxContents
PEXRDTPatternTableContents

PEXRDTTextFontTableContents

The bits in the name sets bitmask are accessed with the following constants:

PEXRDNHighlightNameSet
PEXRDNInvisibilityNameSet
PEXRDNPickNameSet
PEXRDNHighlightNameSetContents
PEXRDNInvisibilityNameSetContents
PEXRDNPickNameSetContents

The bits in the attributes bitmask are accessed with the following constants:

PEXRDAHLHSRMode
PEXRDANPCSubVolume
PEXRDAViewport
PEXRDAClipList
PEXRDAEchoMode

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

See Also

PEXCreateRenderer, **PEXChangeRenderer**, **PEXBeginRendering**

PEXRenderElements - Render Elements

Synopsis

void PEXRenderElements(Display **display*, PEXRenderer *renderer*, PEXStructure *structure*, int *whence1*, long *offset1*, int *whence2*, long *offset2*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of a renderer resource.
structure	The resource identifier of a structure resource.
whence1	A value specifying, with offset1, the first limit of the element range (**PEXBeginning**, **PEXCurrent PEXEnd**).
offset1	An offset from whence1 specifying the first limit of the element range.
whence2	A value specifying, with offset2, the second limit of the element range (**PEXBeginning**, **PEXCurrent PEXEnd**).
offset2	An offset from whence2 specifying the second limit of the element range.

Returns

None

Description

This function processes all output commands in the specified element range of the specified structure. Output primitives in structure are rendered using the specified renderer. If the renderer is not rendering or picking, the request will be ignored. Structures referenced through execute structure output commands are also processed.

The first limit of the range is defined by whence1 and offset1 and the second limit of the range is defined by whence2 and offset2. The whence values describe how to interpret the corresponding offset. **PEXBeginning** means the element position is the value of offset (i.e. the offset from the beginning of the structure). **PEXCurrent** means the element position is the value of the current element pointer position plus the value of offset (i.e. the offset from the current element pointer). **PEXEnd** means the element position is the value of the last element position in the structure plus the value of offset (i.e. the offset from the end of the structure). Offsets can be negative values. If after computing an element position, it is less than zero, the position will be set to zero. If after computing an element position, it is greater than the number of elements in the structure, it will be set to the last structure element in the structure.

Data Structures

```
typedef XID             PEXRenderer;
typedef XID             PEXStructure;
```

Errors

BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for whence was invalid.

See Also

PEXBeginRendering, **PEXCreateStructure**, **PEXCreateRenderer**

PEXRenderNetwork - Render Network

Synopsis

void PEXRenderNetwork(Display **display*, Drawable *drawable*, PEXRenderer *renderer*, PEXStructure *structure*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
renderer	The resource identifier of a renderer resource.
structure	The resource identifier of a structure resource.

Returns

None

Description

This function processes all output commands stored in structure using the specified renderer. Output primitives in structure are rendered to the specified drawable. Structures referenced through execute structure output commands are also processed. This request effectively performs an implicit **PEXBeginRendering** before the traversal of the specified structure network and an implicit **PEXEndRendering**, with a flush value of **True**, after the traversal.

Data Structures

```
typedef XID             PEXRenderer;
typedef XID             PEXStructure;
```

Errors

BadAlloc	The server failed to allocate resources necessary to complete request.
BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported, or the specified renderer resource was not created with a compatible drawable.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The specified renderer was in an invalid state.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXExecuteStructure, **PEXBeginRendering**, **PEXEndRendering**, **PEXCreateStructure**, **PEXCreateRenderer**

Chapter 6

Renderer Picking

The semantics of renderer picking are identical to rendering except that during picking, primitives are hit tested instead of coverted to pixels. All primitives which are passed to the rendering pipeline (i.e. they are not made invisible by the invisibility filter and they are not culled by the current culling mode) are eligible for picking. Picking uses the geometric definition of primitives, so it is possible for hollow, empty and transparent fill areas to be picked. However, it is implementation-dependent whether rendering effects which increase the area of the primitive (such as linewidth) are considered.

Two types of hit test rendering and traversals are supported "pick one" and "pick all". "Pick one" will return at most one primitive which satisfies the pick criteria. "Pick all" will return all hit primitives which satisfy the pick criteria, up to the application specified maximum number of hits. For both types, the hit test which is performed on the primitives is controlled by the specified pick device type, pick inclusion and pick exclusion attributes.

The supported pick device types are inquirable via **PEXGetEnumTypeInfo**. **PEXPickDeviceDCHitBox** is specified by a pick position and a pick distance, both in device coordinates. The pick position is the device coordinate center of the pick area and the shape of the pick area is implementation-dependent (e.g. could be square, rectangular or circular). The pick distance defines the half-width or radius of the pick area. **PEXPickDeviceNPCHitVolume** is specified by two points in normalized projection coordinates. Primitives that lie within or intersect the pick area or volume are considered "hit".

The pick method determines which "hit" primitives will be picked. For "pick one", four standard pick methods are defined: **PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny** and **PEXPickVisibleClosest**. For "pick all", two standard pick methods are defined: **PEXPickAllAll** and **PEXPickAllVisible**. The supported pick methods can be inquired by calling **PEXGetEnumTypeInfo**. PEX servers are required to support at least **PEXPickLast** and **PEXPickAllAll**.

The pick inclusion and exclusion filters specify the name sets to be used to filter primitives after a hit test rendering or traversal. The pick functions will only return primitives which are not excluded by the pick filter. If possible, pick one will return a primitive which most closely satisfies the pick criteria and also passes the pick filter test. A flag is returned from pick one functions to indicate whether a primitive which did not satisfy the pick filter was a better candidate with the specified pick method. The dynamics of the pick inclusion and exclusion filters can be determined by calling **PEXGetRendererDynamics**.

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

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

During a "pick all", multiple primitives may be picked. If the pick method is **PEXPickAllAll**, the primitives (up to the maximum number of hits) which lie within or intersect the hit box are returned. If the pick method is **PEXPickAllVisible**, the primitives (up to the maximum number of hits) which are visible (taking into account the current HLHSR mode) and lie within or intersect the hit box are returned. A "pick first" can be accomplished by specifying the maximum number of hits to be one.

"Pick all" is further controlled by the renderer's pick start path. The pick start path indicates where to begin the next "pick all" and is bound at the start of the "pick all" rendering or traversal.

If a drawable that is associated with a renderer is destroyed while the renderer is performing a "pick one" or "pick all" hit test, an implicit **PEXEndPickOne** or **PEXEndPickAll** (whichever is appropriate) is performed by the server in order to return the renderer to the **PEXIdle** state. All subsequent output and traversal requests to the renderer are ignored and no error is generated.

If a drawable that is associated with a renderer is destroyed or resized while the renderer is performing a "pick one" or "pick all" hit test, the pick operation is terminated and the renderer state is set to **PEXIdle.** All subsequent output and traversal requests to the renderer are ignored until the next **PEXEndPickOne** or **PEXEndPickAll** (whichever is appropriate) which will return a pick status of **PEXAbortedPick** along with an empty pick path.

If a drawable that is associated with a renderer is moved, exposed or occluded while the renderer is performing a "pick one" or "pick all" hit test, the server will continue to process output commands and requests using the new drawable attributes until the pick operation is explicitly or implicitly terminated.

PEXBeginPickAll - Begin Renderer Pick All

Synopsis

void PEXBeginPickAll(Display **display*, Drawable *drawable*, PEXRenderer *renderer*, long *structure_id*, int *method*, int *send_event*, int *max_hits*, int *pick_device_type*, PEXPickRecord **pick_record*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
renderer	The resource identifier of the renderer.
structure_id	A value to be used as the structure identifier for the root of the structure network.
method	The pick all method (**PEXPickAllAll** or **PEXPickAllVisible**).
send_event	**True** or **False** - specifying whether the server should send an event when the maximum number of hits is reached.
max_hits	The maximum number of hits to be returned.
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record.

Returns

None

Description

This functions starts an immediate-mode pick, setting the renderer's renderer state to **PEXPicking**. When the renderer state is **PEXPicking**, primitives are hit tested instead of converted to pixels. All picked primitives are recorded until reaching the maximum hits specified is reached. Additional picked primitives will not be recorded. Once the the maximum number of hits is reached, subsequent primitives may be ignored.

The supported pick device types are inquirable via **PEXGetEnumTypeInfo**. The specified structure identifier will be inserted as the first structure component in the returned pick path(s).

If the send event flag is **True**, and the pick method is **PEXPickAllAll**, then a **PEXMaxHitsReached** event is sent from the server to the client whenever the maximum number of hits is reached by the server, if the event is supported (see **PEXGetImpDepConstants**). Upon receiving the event, the application should stop sending primitives and process the recorded hits. If the pick method is **PEXPickAllVisible**, a complete set of primitives must be sent to the server before determining which primitives are picked.

If the specified drawable does not have the same root and depth as the drawable used to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRenderingTargets**, a match error is generated. If the renderer state is set to **PEXRendering** or **PEXPicking** when this function is called, then the operation in progress is aborted, the **PEXBeginPickAll** function is completed, and a **BadPEXRendererState** error returned.

All functions which process output commands or manipulate attributes (i.e. all output command functions, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**) can be called when the renderer state is **PEXPicking**. They will have the same semantics except that primitives are hit tested instead of converted to pixels.

Data Structures

```
typedef XID             PEXRenderer;

typedef union {
    PEXPDNPCHitVolume       volume;
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;

typedef struct {
    int             type;
    unsigned long   serial;     /* # of last request processed by server */
    Bool            send_event; /* true if this came from a SendEvent request */
    Display         *display;   /* Display the event was read from */
    PEXRenderer     renderer;
} PEXMaxHitsReachedEvent;
```

Errors

BadAlloc	The server failed to allocate resources necessary to complete request.
BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported, or the specified renderer resource was not created with a compatible drawable.
BadPEXRenderer	The specified renderer resource identifier is invalid.

BadPEXRendererState
The specified renderer was in an invalid state.

BadValue The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXEndPickAll, **PEXPickAll**, **PEXGetImpDepConstants**

PEXBeginPickOne - Begin Renderer Pick One

Synopsis

void PEXBeginPickOne(Display **display*, Drawable *drawable*, PEXRenderer *renderer*, long *structure_id*, int *method*, int *pick_device_type*, PEXPickRecord **pick_record*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
renderer	The resource identifier of the renderer.
structure_id	A value to be used as the structure identifier for the root of the structure network.
method	The pick one method (**PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny**, **PEXPickVisibleClosest**).
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record.

Returns

None

Description

This functions starts an immediate-mode pick, setting the renderer's renderer state to **PEXPicking**. When the renderer state is **PEXPicking**, primitives will be hit tested instead of converted to pixels. For "pick one", a hierarchical path to the picked primitive will be maintained.

Standard "pick one" methods are **PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny** and **PEXPickVisibleClosest**. The supported pick device types are inquirable via **PEXGetEnumTypeInfo**. The specified structure identifier will be inserted as the first structure component in the returned pick path.

If the specified drawable does not have the same root and depth as the drawable that was used to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRenderingTargets**, a Match error will be generated. If the renderer state is set to **PEXRendering** or **PEXPicking** when this function is called, then the operation in progress is aborted, the **PEXBeginPickOne** function is completed, and a **BadPEXRendererState** error is sent.

All functions which process output commands or manipulate attributes (i.e. all output command functions, **PEXBeginStructure**, **PEXEndStructure**, **PEXRenderElements**, and **PEXAccumulateState**) can be called when the renderer state is **PEXPicking**. They will have the same semantics except that primitives are hit tested instead of converted to pixels.

Data Structures

```
typedef XID             PEXRenderer;

typedef union {
    PEXPDNPCHitVolume       volume;
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
```

```
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;
```

Errors

BadAlloc The server failed to allocate resources necessary to complete request.

BadDrawable The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported, or the specified renderer resource was not created with a compatible drawable.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXRendererState
The specified renderer was in an invalid state.

BadValue The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXEndPickOne, **PEXPickOne**

PEXFreePickPaths - Free Memory Allocated for Pick Paths

Synopsis

void PEXFreePickPaths(unsigned long *count*, PEXPickPath **pick_paths*)

Arguments

count	The number of pick paths.
pick_paths	An array of pick paths.

Returns

None

Description

This function deallocates memory returned by **PEXEndPickAll** and **PEXPickAll**.

Errors

None

See Also

PEXEndPickAll, **PEXPickAll**

PEXEndPickAll - End Pick All

Synopsis

PEXPickPath *PEXEndPickAll(Display **display*, PEXRenderer *renderer*, int **status_return*, int **more_return*, unsigned long **count_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of the renderer.
status_return	Returns the status of the pick operation.
more_return	Returns the status of remaining picks.
count_return	Returns the number of pick paths.

Returns

An array of pick paths; a null pointer if unsuccessful or no pick (see also status_return).

Description

This function terminates an immediate-mode pick, returns the hierarchical pick paths of any hit primitives, and sets the renderer state to **PEXIdle**.

If one or more primitives were picked, a pick status of **PEXPick** is returned along with the pick paths. The hierarchical pick path is equivalent to the renderer's current path at the time the picked primitive was processed. If no primitives were picked, the returned pick status is **PEXNoPick**, and the returned pick paths is a null pointer. If the renderer's drawable was destroyed or resized during the pick operation, the returned pick status is **PEXAbortedPick** and the returned pick paths is a null pointer.

If all hits were recorded then **PEXNoMoreHits** is returned and the renderer's pick start path will be empty. If the maximum number of hits was reached and additional hits were detected, then **PEXMoreHits** is returned and the renderer's pick start path will be set to the last recorded hit primitive. If, after reaching the maximum number of hits, subsequent output commands were ignored, then **PEXMayBeMoreHits** is returned and the renderer's pick start path is set to the last element processed by the renderer before it started ignoring primitives.

If the renderer state is **PEXIdle** when this function is called, (i.e., no picking is in progress or the rendering was aborted due to a resize), the function is ignored and no error is generated. If the renderer state is currently **PEXRendering** or if the pick operation in progress is a "pick one", then a **BadPEXRendererState** error is sent.

PEXlib allocates memory for the return value. **PEXFreePickPaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXRenderer;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
```

```
} PEXPickElementRef;

typedef XID             PEXStructure;
```

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXRendererState
The specified renderer was in an invalid state.

See Also

PEXBeginPickAll, **PEXPickAll**, **PEXFreePickPaths**

PEXEndPickOne - End Pick One

Synopsis

PEXPickPath *PEXEndPickOne(Display **display*, PEXRenderer *renderer*, int **status_return*, int **undetectable_return*)

Arguments

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

renderer The resource identifier of the renderer.

status_return Returns the status of the pick operation.

undetectable_return
Returns **True** or **False** indicating whether another pick better satisfied the pick criteria with the exception that it did not pass the pick filter test.

Returns

A pointer to the pick path; a null pointer if unsuccessful or no pick (see also status_return).

Description

This function terminates an immediate-mode pick, returns the hierarchical pick path to the closest or last hit primitive, and sets the renderer state to **PEXIdle**.

If a primitive was picked, the returned pick status is **PEXPick**. If no primitive was picked, the returned pick status is **PEXNoPick**, and the returned pick path is a null pointer. If the renderer's drawable was destroyed or resized during the pick operation, the returned pick status is **PEXAbortedPick** and the returned pick path is a null pointer.

If there was a primitive which more closely satisfied the pick criteria, but did not pass the pick filter test, then the undetectable pick return status will be **True**. Otherwise, it will be **False**.

If the renderer state is currently **PEXIdle** when this function is called, (i.e., no picking is in progress or the rendering was aborted due to a resize), the function is ignored and no error is generated. If the renderer state is currently **PEXRendering** or if the pick operation in progress is a "pick all", then a **BadPEXRendererState** error is sent.

PEXlib allocates memory for the return value. **PEXFreePickPaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXRenderer;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
} PEXPickElementRef;

typedef XID             PEXStructure;
```

Errors

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXRendererState
The specified renderer was in an invalid state.

See Also

PEXBeginPickOne, **PEXPickOne**,

PEXPickAll - Pick All Traversal

Synopsis

PEXPickPath *PEXPickAll(Display **display*, Drawable *drawable*, PEXRenderer *renderer*, int *method*, int *max_hits*, int *pick_device_type*, PEXPickRecord **pick_record*, int **status_return*, int **more_return*, unsigned long **count_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
renderer	The resource identifier of the renderer.
method	The pick all method (**PEXPickAllAll** or **PEXPickAllVisible**).
max_hits	The maximum number of hits to be returned.
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record.
status_return	Returns the status of the pick operation.
more_return	Returns the status of remaining picks.
count_return	Returns the number of pick paths.

Returns

An array of pick paths; a null pointer if unsuccessful or no pick (see also status_return).

Description

This function traverses the structure network specified by the renderer's current pick start path. Hit testing begins after the last element specified in the renderer's current pick start path and continues until one of two conditions is met: the maximum number of hits is reached, or the last element of the first structure in the pick start path is processed. If the pick start path does not define a valid hierarchical path, a **BadPEXPath** error is sent and a null pick path is returned.

Standard "pick all" methods are **PEXPickAllAll** and **PEXPickAllVisible**. The supported pick device types are inquirable via **PEXGetEnumTypeInfo**.

If one or more primitives were picked, a pick status of **PEXPick** is returned along with the pick paths. The hierarchical pick path is equivalent to the renderer's current path at the time the picked primitive was processed. If no primitives were picked, the returned pick status is **PEXNoPick**, and the returned pick paths is a null pointer. If the renderer's drawable was destroyed or resized during the pick operation, the returned pick status is **PEXAbortedPick** and the returned pick paths is a null pointer.

The paths of all hit primitives are recorded until reaching the maximum number of hits or until the server maximum number of recordable hits is reached. Once the maximum number of paths is recorded, subsequent primitives may be ignored and the results returned.

If all possible hits were recorded then **PEXNoMoreHits** is returned and the renderer's pick start path will be empty. If the maximum number of hits was reached and additional hits were detected, then **PEXMoreHits** is returned and the renderer's pick start path will be set to the last recorded hit primitive. If, after reaching the maximum number of hits, subsequent output commands were ignored, then **PEXMayBeMoreHits** is returned and the renderer's pick start path will be set to the last element processed by the renderer before it started ignoring primitives.

If the specified drawable does not have the same root and depth as the drawable used to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRenderingTargets**, a match error is generated. If the renderer state is set to **PEXRendering** or **PEXPicking** when this function is called, then the operation in progress is aborted, the **PEXPickAll** function is completed, and a **BadPEXRendererState** error returned.

PEXlib allocates memory for the return value. **PEXFreePickPaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXRenderer;

typedef union {
    PEXPDNPCHitVolume       volume;
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
} PEXPickElementRef;

typedef XID             PEXStructure;
```

Errors

BadAlloc	The server failed to allocate resources necessary to complete request.
BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported, or the specified renderer resource was not created with a compatible drawable.
BadPEXRenderer	The specified renderer resource identifier is invalid.
BadPEXRendererState	The specified renderer was in an invalid state.
BadPEXPath	The renderer pick start path is invalid.
BadValue	The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXBeginPickAll, **PEXEndPickAll**, **PEXFreePickPaths**

PEXPickOne - Pick One Traversal

Synopsis

PEXPickPath *PEXPickOne(Display **display*, Drawable *drawable*, PEXRenderer *renderer*, PEXStructure *structure*, int *method*, int *pick_device_type*, PEXPickRecord **pick_record*, int **status_return*, int **undetectable_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
renderer	The resource identifier of the renderer.
structure	The resource identifier for the root structure of the structure network.
method	The pick one method (**PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny** **PEXPickVisibleClosest**).
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
pick_record	A pointer to the pick data record.
status_return	Returns the status of the pick operation.
undetectable_return	Returns **True** or **False** indicating whether another pick better satisfied the pick criteria with the exception that it did not pass the pick filter test.

Returns

A pointer to the pick path; a null pointer if unsuccessful or no pick (see also status_return).

Description

This function traverses the specified structure network.

Standard "pick one" methods are **PEXPickLast**, **PEXPickClosestZ**, **PEXPickVisibleAny** and **PEXPickVisibleClosest**. The supported pick device types are inquirable via **PEXGetEnumTypeInfo**.

If a primitive was picked, the returned pick status is **PEXPick**. If no primitive was picked, the returned pick status is **PEXNoPick**, and the returned pick path is a null pointer. If the renderer's drawable was destroyed or resized during the pick operation, the returned pick status is **PEXAbortedPick** and the returned pick path is a null pointer.

If there was a primitive which more closely satisfied the pick criteria, but did not pass the pick filter test, then the undetectable pick return status will be **True**. Otherwise, it will be **False**.

If the specified drawable does not have the same root and depth as the drawable that was used to create the renderer, or, if the specified drawable is not one of the supported drawables returned by **PEXMatchRenderingTargets**, a Match error will be generated. If the renderer state is set to **PEXRendering** or **PEXPicking** when this function is called, then the operation in progress is aborted, the **PEXPickOne** function is completed, and a **BadPEXRendererState** error is sent.

PEXlib allocates memory for the return value. **PEXFreePickPaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXRenderer;

typedef union {
    PEXPDNPCHitVolume      volume;
```

```
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
} PEXPickElementRef;

typedef XID             PEXStructure;
```

Errors

BadAlloc The server failed to allocate resources necessary to complete request.

BadDrawable The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported, or the specified renderer resource was not created with a compatible drawable.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXRendererState
The specified renderer was in an invalid state.

BadPEXStructure The specified structure resource identifier is invalid.

BadValue The pick record contains invalid data, or the pick device type is invalid.

See Also

PEXBeginPickOne, **PEXEndPickOne**

Chapter 7

Structures

A structure is a resource which stores output commands for later execution. Output commands stored in a structure are referred to as structure elements. Structures can be shared by clients like any other X or PEX resource. Structures can reference other structures; such a hierarchy of structures is called a structure network. The first structure in a structure network is called the root structure.

The two modifiable attributes in a structure resource are:

element pointer
editing mode

The following attributes of a structure can be inquired:

number of structure elements
length (in units of four bytes)
whether or not referenced by other structures

Create a structure resource by calling the **PEXCreateStructure** function. Add structure elements to existing structures simply by calling output command functions with the appropriate structure identifier and a **PEXOCRequestType** of **PEXOCStore**.

Delete structure resources by calling the **PEXDestroyStructures** function. **PEXDestroyStructures** takes a list of structure resources to be deleted and has the side effect of deleting all other structure elements which reference any of the deleted structures.

The position of an element within a structure is its offset from the start of the structure. The first element of a structure has an offset of one. The last element of a structure has an offset of n, where n is the total number of elements within the structure. Conceptually, a "null element" exists before the first element and has an offset of zero.

The element pointer contains an offset into the structure. When a structure is first created, the element pointer has a value of zero. Many commands use the element pointer value to fetch, store, or edit structure elements. PEXlib contains a number of functions which modify the position of the element pointer. These requests typically require a "{whence, offset}" pair. The element pointer is positioned relative to the beginning of the structure (whence == **PEXBeginning**), relative to the current element pointer position (whence == **PEXCurrent**), or relative to the end of the structure (whence == **PEXEnd**). To obtain the absolute offset, the specified offset is added to the position indicated by "whence". For functions which specify a range of elements with two "{whence, offset}" pairs, the elements affected are those between the two computed positions regardless of which position was specified first and which was second.

A label is a special structure element that allows convenient editing of a structure. Labels are stored as structure elements and move with other structure elements as elements are inserted or deleted before them in the structure. The element pointer can be moved to a particular label or to the next occurrence of a particular structure element within the structure.

An output command is inserted into a structure by setting the structure editing mode to **PEXStructureInsert** (the default), setting the element pointer to the desired element offset, and calling the appropriate output command functions. In insert mode, each output command sent to a structure causes a structure element to be created and inserted after the element specified by the element pointer. After each insertion, the element pointer is updated to point to the newly-inserted element.

To illustrate a common case, the element pointer points to the last element in a structure and structure elements are to be added to the end of the structure. Each output command sent to this structure causes a structure element to be added after the last element, and the element pointer is then updated to point to the newly-inserted structure element. Therefore, multiple output commands sent to a structure when the editing mode is **PEXStructureInsert** result in an ordered list of structure elements being added to the structure.

Existing structure elements are overwritten by setting the editing mode of a structure to **PEXStructureReplace**. In replace mode, existing elements are deleted and the new elements sent to the structure are created in their place. The number of elements deleted is equal to n, where n is the number of elements sent to the structure in a single request (unless there are fewer than n elements between the element pointer position and the end of the structure, in which case only those between the element pointer position and the end of the structure are deleted). If the range to be deleted includes offset 0 then n-1 elements are deleted from the structure. The elements sent to the structure are then inserted in place of the deleted elements. See also the introduction in *Chapter 2: Output Commands*.

It is not possible to set the element pointer to an offset greater than the last element in the structure. This means at least one element is always overwritten during a replace operation, unless the structure originally contained no elements. The element pointer is left pointing to the last inserted or replaced element in the structure.

Some structure editing commands can copy or delete a range of structure elements. When elements are deleted, the element pointer is set to the position immediately preceding the first element deleted. The offset of each element after those deleted decreases by one for each element deleted. Deleting element zero is effectively a no-op. A deletion operation includes the elements at the boundaries of the deletion range. After the deletion occurs, the element pointer is left pointing at the element preceding the deleted range.

Copy operations ignore the editing mode of the destination structure; elements are always inserted during copy operations. After a copy operation, the destination structure's element pointer is left pointing at the last copied element. Copying elements when the source and destination structure are the same is also permitted. The semantics specify that the copy occurs as though the elements in the copy range are first copied to a temporary area and then inserted at the desired spot in the destination structure.

Other commands exist for dealing with structures. These include commands to change all references to a structure, obtain the size of a structure, and obtain information about the hierarchy of a structure network. Some of these requests use the concept of an element reference path, which is an arbitrary length list of structure id/element offset pairs. A structure element is referenced with such a structure id/element offset pair. Since structures can call other structures, it is helpful to know the organization of the structure network. The element reference path is used to identify a unique path through a structure network.

For instance, if structure "A" is the root of a structure network, an element that lives at element offset 14 in structure "A" has an element reference path equal to {"A":14}. If element 14 in structure "A" is an execute structure command that references structure "B," element zero in "B" has an element reference path of {"A":14, "B":0}. Since it is possible that "B" is also referenced by other structures, there may be alternative paths to the element "B":0. The depth of an element reference path is the number of element references in the path. A path depth of zero requests all unique paths leading to a particular element.

PEXChangeStructureRefs - Change Structure References

Synopsis

void PEXChangeStructureRefs(Display **display*, PEXStructure *old_structure*, PEXStructure *new_structure*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
old_structure	The resource identifier of the structure no longer to be referenced.
new_structure	The resource identifier of the structure now referenced.

Returns

None

Description

This function changes "execute structure" elements in the server that reference the specified old structure into "execute structure" elements which reference the specified new structure. Both structures must already exist as valid structure resources.

Any references to the new structure that existed before this request are not affected. If there were references to the old structure and the new structure does not exist, an error is returned and no action is taken.

On all PHIGS workstation resources where the new structure is not already posted and the old structure is posted, the new structure is posted with the same priority as the old structure and the old structure is unposted.

Data Structures

```
typedef XID              PEXStructure;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateStructure PEXGetStructureInfo, **PEXGetStructuresInNetwork**, **PEXGetAncestors**, **PEXGetDescendants**

PEXCopyElements - Copy Elements

Synopsis

void PEXCopyElements(Display **display*, PEXStructure *src_structure*, int *src_whence1*, long *src_offset1*, int *src_whence2*, long *src_offset2*, PEXStructure *dst_structure*, int *dst_whence*, long *dst_offset*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
src_structure	The resource identifier of the source structure.
src_whence1	A value specifying, with src_offset1, the first limit of the range of elements to be copied (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
src_offset1	The offset from src_whence1 denoting the first limit of the range of elements to be copied.
src_whence2	A value specifying, with src_offset2, the second limit of the range of elements to be copied (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
src_offset2	The offset from src_whence2 denoting the second limit of the range of elements to be copied.
dst_structure	The resource identifier of the destination structure.
dst_whence	A value specifying, with dst_offset, the position at which the elements are inserted into the destination structure (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
dst_offset	The offset from dst_whence denoting the position at which the elements are inserted into the destination structure.

Returns

None

Description

This function copies a range of elements from the specified source structure to the specified destination structure.

If a computed offset is less than zero, it is set to zero before obtaining the element information. If a computed offset is greater than the number of elements in the structure, the offset is set to the offset of the last element in the structure.

The source structure and destination structure can be the same. In this case, the copy operation proceeds as though the indicated range were copied to a temporary location and then inserted relative to the destination position.

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

Data Structures

```
typedef XID            PEXStructure;
```

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for whence parameter is invalid.

See Also

PEXCreateStructure

PEXCopyStructure - Copy Structure

Synopsis

void PEXCopyStructure(Display **display*, PEXStructure *src_structure*, PEXStructure *dst_structure*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
src_structure	The resource identifier of the source structure.
dst_structure	The resource identifier of the destination structure.

Returns

None

Description

This function copies elements in the source structure to the destination structure. Both structures must already exist as valid structure resources. Elements already in the destination structure are overwritten. The element pointer and editing mode attributes of the source structure are copied to the destination structure as well.

Data Structures

```
typedef XID              PEXStructure;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateStructure

PEXCreateStructure - Create Structure

Synopsis

PEXStructure PEXCreateStructure(Display **display*)

Argument

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

Returns

The resource identifier of the newly-created structure resource.

Description

This function creates a structure resource and returns the resource identifier of the created structure. The returned identifier is used to refer to the created structure resource.

Data Structures

```
typedef XID             PEXStructure;
```

Errors

BadAlloc The server failed to allocate the resource.

See Also

PEXDestroyStructures

PEXDeleteBetweenLabels - Delete Elements Between Labels

Synopsis

void PEXDeleteBetweenLabels(Display **display*, PEXStructure *structure*, long *label1*, long *label2*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
label1	The first label.
label2	The second label.

Returns

None

Description

This function deletes a range of elements from the specified structure. Elements between the two labels are deleted. The label elements are not deleted. A search for the first label is performed starting at the current offset plus one. A search for the second label is performed starting at the element following the first label. After the deletion operation, the structure element pointer is set to the pointer position at the first label.

If either of the two labels is not found between the starting point of the search and the end of the structure, no deletion occurs and the structure's element point is left unchanged.

Data Structures

```
typedef XID              PEXStructure;
```

Errors

BadPEXLabel	The specified label does not exist.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXCreateStructure, **PEXLabel**

PEXDeleteElements - Delete Elements

Synopsis

void PEXDeleteElements(Display **display*, PEXStructure *structure*, int *whence1*, long *offset1*, int *whence2*, long *offset2*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
whence1	A value specifying, with offset1, the first limit of the range of elements to be deleted (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
offset1	The offset from whence1 denoting the first limit of the range of elements to be deleted.
whence2	A value specifying, with offset2, the second limit of the range of elements to be deleted (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
offset2	The offset from whence2 denoting the second limit of the range of elements to be deleted.

Returns

None

Description

This function deletes a range of elements from the structure specified by structure.

If a computed offset is less than zero, it is set to zero before obtaining the element information. If a computed offset is greater than the number of elements in the structure, the offset is set to the offset of the last structure element in the structure. Deleting the null element is effectively a no-op. After the deletion operation, the structure element pointer is set to the element immediately preceding the range of deleted elements.

Data Structures

```
typedef XID             PEXStructure;
```

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for whence parameter is invalid.

See Also

PEXCreateStructure

PEXDeleteToLabel - Delete Elements to Label

Synopsis

void PEXDeleteToLabel(Display **display*, PEXStructure *structure*, int *whence*, long *offset*, long *label*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
whence	A value specifying, with offset, the beginning of the range of elements to be deleted (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
offset	The offset from whence denoting the beginning of the range of elements to be deleted.
label	The label specifying the end of the range of elements to be deleted.

Returns

None

Description

This function deletes a range of elements between a computed offset and a specified label in the specified structure. The computed offset specifies the beginning of the deletion range. The label specifies the end of the deletion range. Elements are deleted from the structure element immediately after the computed offset up to the next occurrence of the label. The label is not deleted. If label is not found, no elements are deleted.

If the computed offset is less than zero, it is set to zero before the deletion occurs. If the computed offset is greater than the number of elements in the structure, the offset is set to the offset of the last structure element. Deleting the zero element is effectively a no-op. After the deletion operation, the structure element pointer is set to the element immediately preceding the range of deleted elements.

Data Structures

```
typedef XID             PEXStructure;
```

Errors

BadPEXLabel	The specified label does not exist.
BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for whence parameter is invalid.

See Also

PEXCreateStructure, **PEXLabel**

PEXDestroyStructures - Destroy Structures

Synopsis

void PEXDestroyStructures(Display **display*, unsigned long *count*, PEXStructure **structures*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
count	The number of structure resource identifiers.
structures	An array of resource identifiers of the structures to be destroyed.

Returns

None

Description

This function deletes each structure in the list of structure identifiers, removes all references to it in the server, and frees all memory associated with it.

This function also removes any "execute structure" structure elements that reference a structure in the list and unposts any structure in the list from a PHIGS workstation resource to which it is posted. If a structure has any structure elements removed from it as a result of this call, its element pointer will continue to point at the same structure element. However, if the structure element being pointed at is removed, the element pointer will be positioned at the previous structure element.

Any paths in search contexts or pick measures that contain references to a deleted structure may still be inquired. However, if a path in a search context or pick measure resource which contains a destroyed structure resource identifier is later used in a **PEXSearchNetwork** or **PEXUpdatePickMeasure** function, a path error is generated.

Data Structures

```
typedef XID              PEXStructure;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXCreateStructure, **PEXGetStructuresInNetwork**, **PEXGetAncestors**, **PEXGetDescendants**, **PEXChangeStructureRefs**

PEXElementSearch - Element Search

Synopsis

Status PEXElementSearch(Display **display*, PEXStructure *structure*, int *whence*, long *offset*, int *direction*, unsigned long *incl_count*, unsigned short **incl_list*, unsigned long *excl_count*, unsigned short **excl_list*, unsigned long **elem_offset_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
whence	A value specifying, with offset, the offset at which the search is to begin (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
offset	The offset from whence at which the search is to begin.
direction	The direction of the search (**PEXForward** or **PEXBackward**).
incl_count	The number of values in inclusion array.
incl_list	An array of short integers specifying structure elements to be searched for.
excl_count	The number of values in exclusion array.
excl_list	An array of short integers specifying structure elements not to be searched for.
elem_offset_return	Returns the offset of the element located by the search.

Returns

Zero if unsuccessful, non-zero otherwise. The non-zero value will be either **PEXFound** or **PEXNotFound** depending upon the result of the search.

Description

This function searches for the first occurrence of the specified element type in the specified structure. The search always includes the starting element.

If the computed offset is less than zero, it is set to zero before the search is performed. If the 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.

An element is selected if its element type is contained in inclusion list and is not contained in exclusion list. An element type of **PEXOCAll** causes all element types to match. If a structure element type is in both the inclusion and exclusion list, it is excluded.

The search terminates if a match is found or if the limits of the structure are reached. The search progresses from the start point in the specified direction (**PEXForward** or **PEXBackward**). This is a non-descending search; that is, the search does not include any structures referenced by "execute structure" elements. If the search finds a match, a return status of **PEXFound** and the offset of the matching element is returned. If the search is unsuccessful, a return status of **PEXNotFound** is returned.

The element pointer position of structure is not changed.

Data Structures

```
typedef XID             PEXStructure;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

BadValue The specified value for whence or direction parameter is invalid.

See Also

PEXCreateStructure

PEXFetchElements - Fetch Elements

Synopsis

Status PEXFetchElements(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*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
whence1	A value 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 (structure elements).

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.

An null pointer is returned is 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 The specified value for whence parameter is invalid.

See Also

PEXCreateStructure
PEXDecodeOCs, **PEXEncodeOCs**, **PEXSendOCs**

PEXFetchElementsAndSend - Fetch Elements and Send to Display

Synopsis

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

Arguments

src_display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
whence1	A value 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 **PEXFetchElements** 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 **PEXFetchElements**, 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, the 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 type 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

```
typedef XID              PEXStructure;
```

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for whence parameter is invalid.

See Also

PEXFetchElements

PEXFreeStructurePaths - Free Structure Paths Memory

Synopsis

void PEXFreeStructurePaths(unsigned long *count*, PEXStructurePath **paths*)

Arguments

count	The number of structure paths.
paths	An array of structure paths.

Returns

None

Description

This function deallocates memory returned by **PEXGetAncestors**, **PEXGetDescendants**, and **PEXSearchNetwork**.

Errors

None

See Also

PEXGetAncestors, **PEXGetDescendants**, **PEXSearchNetwork**.

PEXGetAncestors - Get Ancestors

Synopsis

PEXStructurePath *PEXGetAncestors(Display **display*, PEXStructure *structure*, int *path_part*, unsigned long *path_depth*, unsigned long **count_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
path_part	The part of the path to return (**PEXTopPart** or **PEXBottomPart**).
path_depth	The maximum number of structure network path levels to be returned in each path found.
count_return	Returns the number of paths found.

Returns

An array of structure paths defining the ancestors of the specified structure; a null pointer if unsuccessful.

Description

This function returns a list of structure network paths which reference the specified structure. Paths are returned as lists of element references, each of which is represented as a structure resource identifier and an offset that gives the element's position in the structure. Only unique paths are returned; in other words there will be no duplicates in the list of returned paths.

The path part must be either **PEXTopPart**, which requests that the top of the structure paths be returned, or **PEXBottomPart**, which requests that the bottom of the structure paths be returned.

The path depth specifies the maximum number of element references to be returned in each path. If the path depth is 0, the entire path is returned. If the path depth is 1, only one element reference is returned for each path. A path depth of 2 returns two elements, and so on.

Specifying a path depth of 0 and a path part of **PEXTopPart** returns the unique top parts of all paths to structure. Specifying a path depth of 1 and a path part of **PEXTopPart** returns the root structure of all structure networks which contain structure. A path depth of 2 and path part of **PEXBottomPart** returns all of the structure's immediate ancestors. Determine the number of references to structure by setting the path depth to 1 and the path part to **PEXBottomPart**.

PEXlib allocates memory for the returned ancestor information. **PEXFreeStructurePaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXStructure;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;

typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

BadValue The specified value for path part is invalid.

See Also

PEXCreateStructure, **PEXGetStructuresInNetwork**, **PEXGetDescendants**

PEXGetDescendants - Get Descendants

Synopsis

PEXStructurePath *PEXGetDescendants(Display **display*, PEXStructure *structure*, int *path_part*, unsigned long *path_depth*, unsigned long **count_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
path_part	The part of the path to return (**PEXTopPart** or **PEXBottomPart**).
path_depth	The maximum number of structure network path levels to be returned in each path found.
count_return	Returns the number of paths found.

Returns

An array of structure paths defining the descendants of the specified structure; a null pointer if unsuccessful.

Description

This function returns a list of structure network paths referenced by the specified structure. The elements of the returned array are of type **PEXStructurePath**. Paths are returned as lists of element references, each of which is represented as a structure resource identifier and an offset that gives the element's position in the structure. Only unique paths are returned; in other words, there will be no duplicates in the returned paths.

The path part must be either **PEXTopPart**, which requests that the top of the structure paths be returned, or **PEXBottomPart**, which requests that the bottom of the structure paths be returned.

The path depth specifies the maximum number of element references to be returned in each path. If the path depth is 0, the entire path is returned. If the path depth is 1, only one element reference is returned for each path, and so on.

For instance, specifying a path depth of 0 and a path part of **PEXTopPart** returns all paths from structure to leaf nodes in the structure network tree in the order they would be traversed. Specifying a path depth of 1 and a path part of **PEXBottomPart** determines the bottom-most structures of the structure network rooted at structure.

PEXlib allocates memory for the returned descendant information. **PEXFreeStructurePaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXStructure;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;

typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

BadValue The specified value for path part is invalid.

See Also

PEXCreateStructure, **PEXGetStructuresInNetwork**, **PEXGetAncestors**

PEXGetElementInfo - Get Element Information

Synopsis

Status PEXGetElementInfo(Display **display*, PEXStructure *structure*, int *whence1*, long *offset1*, int *whence2*, long *offset2*, int *float_format*, unsigned long **count_return*, PEXElementInfo ***info_return*)

Arguments

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

structure The resource identifier of the structure.

whence1 A value specifying, with offset1, the first limit of the range of queried elements (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).

offset1 The offset from whence1 denoting the first limit of the range of queried elements.

whence2 A value specifying, with offset2, the second limit of the range of elements to be queried (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).

offset2 The offset from whence2 denoting the second limit of the range of elements to be queried.

float_format The floating point format to use when computing element sizes (**PEX-IEEE_754_32**, **PEXDEC_F_Floating**, **PEXIEEE_754_64**, **PEXDEC_D_Floating**).

count_return Returns the number element info records returned.

info_return Returns an array of element info records describing the elements in the specified range.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns information about a range of elements from the specified structure.

If a computed offset is less than zero it is set to zero before obtaining the element information. If a 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.

Information returned about the list of inquired elements includes the type of each element and its size. The size of each element is based upon the specified floating point format. No information is returned for inquires on element offset zero. The element pointer is not affected by this function.

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

Data Structures

```
typedef XID             PEXStructure;

typedef struct {
    unsigned short      type;
    unsigned short      length;
} PEXElementInfo;
```

Errors

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXStructure The specified structure resource identifier is invalid.

BadValue The specified value for whence parameter is invalid.

See Also

PEXCreateStructure

PEXGetStructureInfo - Get Structure Information

Synopsis

Status PEXGetStructureInfo(Display **display*, PEXStructure *structure*, int *float_format*, unsigned long *value_mask*, PEXStructureInfo **info_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
float_format	The floating point format to use when computing element sizes (**PEXIEEE_754_32**, **PEXDEC_F_Floating**, **PEXIEEE_754_64**, **PEXDEC_D_Floating**).
value_mask	A mask indicating which values to return.
info_return	Returns information about the structure resource.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns information about the specified structure resource. The value mask is constructed by or'ing together the following constants:

PEXElementPtr
PEXNumElements
PEXLengthStructure
PEXHasRefs
PEXEditMode

The length of the structure is given in the number of 4-byte units, and is based upon the specified floating point format.

Data Structures

```
typedef XID             PEXStructure;

typedef struct {
    unsigned long       element_count;
    unsigned long       size;
    Bool                has_refs;
    unsigned short      edit_mode;
    unsigned long       element_pointer;
} PEXStructureInfo;
```

Errors

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXStructure The specified structure resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreateStructure

PEXGetStructuresInNetwork - Get Structures in Network

Synopsis

PEXStructure *PEXGetStructuresInNetwork(Display **display*, PEXStructure *structure*, int *which*,
unsigned long **count_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the root structure in the structure network.
which	A value indicating which structure resource identifiers to return (**PEXAll** or **PEXOrphans**).
count_return	Returns the number of structure resource identifiers returned.

Returns

An array of structure resource identifiers; a null pointer if unsuccessful.

Description

This function returns a list of unique structure resource identifiers that are referenced in the structure network rooted at the specified structure. **PEXAll** indicates that identifiers of all structure resources referenced in the structure network are returned. **PEXOrphans** indicates that identifiers returned are those not referenced by any structures outside of those in the specified structure network.

The specified structure identifier will always be returned in the list, unless it is an invalid structure identifier, in which case the returned list will empty.

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

Data Structures

```
typedef XID             PEXStructure;
```

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for which parameter is invalid

See Also

PEXCreateStructure, **PEXGetAncestors**, **PEXGetDescendants**

PEXSetEditingMode - Set Structure Editing Mode

Synopsis

void PEXSetEditingMode(Display *display, PEXStructure structure, int mode)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
mode	The editing mode (**PEXStructureInsert** or **PEXStructureReplace**).

Returns

None

Description

This function sets the editing mode for the structure specified. The editing mode specifies how editing operations affect the structure. If the editing mode is **PEXStructureInsert**, subsequent requests to create structure elements cause elements to be inserted into the structure. The element pointer is then incremented by the number of elements inserted. If the editing mode is **PEXStructureReplace**, output requests that create structure elements cause structure elements to replace elements, starting at the location specified by the element pointer.

Data Structures

```
typedef XID              PEXStructure;
```

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for mode is invalid.

See Also

PEXCreateStructure

PEXSetElementPtr - Set Structure Element Pointer

Synopsis

void PEXSetElementPtr(Display **display*, PEXStructure *structure*, int *whence*, long *offset*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
whence	A value specifying with offset, the element pointer position (**PEXBeginning**, **PEXCurrent**, **PEXEnd**).
offset	The offset from whence which specifies the element pointer position.

Returns

None

Description

This function sets the element pointer for the structure specified to the position specified.

If either computed offset is less than zero, it is set to zero before obtaining the element information. 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.

Data Structures

```
typedef XID             PEXStructure;
```

Errors

BadPEXStructure	The specified structure resource identifier is invalid.
BadValue	The specified value for whence parameter is invalid.

See Also

PEXCreateStructure

PEXSetElementPtrAtLabel - Set Structure Element Pointer at Label

Synopsis

void PEXSetElementPtrAtLabel(Display **display*, PEXStructure *structure*, long *label*, long *offset*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
label	The value of the label.
offset	The offset from the label.

Returns

None

Description

This function sets the element pointer for the specified structure at a position denoted by the label. A search is conducted for the next occurrence of the label, starting at the current element pointer position plus one and proceeding in the forward direction. If label is found, the element pointer for the structure is set to the location of the label plus the value of the specified offset. If label is not found, the structure's element pointer is left unchanged.

Data Structures

```
typedef XID              PEXStructure;
```

Errors

BadPEXLabel	The specified label does not exist.
BadPEXStructure	The specified structure resource identifier is invalid.

See Also

PEXCreateStructure, **PEXLabel**

Chapter 8

Name Sets

A name set is a PEX resource which maintains a set of identifiers called names. A name set allows conditional treatment of output primitives during rendering, picking, and structure searching operations.

Search contexts and pick measures also have attributes that are name set resource identifiers.

Renderer resources and PHIGS workstation resources each have a name set attribute considered as the current name set. The contents of the current name set cannot be directly modified or queried in the PHIGS workstation resource. The name set used in a renderer resource is set through the pipeline context resource that provides the initial state for the renderer. There are output commands that cause identifiers to be added to or removed from this name set. Thus, as a structure traversal or series of output commands is executed, the contents of the current name set can fluctuate.

Special filters are built into renderer, search context, pick measure, and PHIGS workstation resources. These filters consist of a pair of name set resource identifiers, one of which is considered as the "inclusion set" and the other which is considered as the "exclusion set". Both are provided because sometimes it is easier to specify what passes the filter test, while other times it is easier to specify what does not pass the filter test.

The filtering criteria is successfully met if, when an output primitive command is executed, the inclusion name set contains at least one name contained in the current name set, and exclusion name set does not contain any names in the current name set. If a name is contained in both the inclusion set and the exclusion set, it is considered to be only in the exclusion set.

Conditional operations that are supported with name sets include highlighting, treating primitives as though they were invisible, and ignoring certain primitives when searching through a structure network or performing a picking operation.

The user can render the scene in different ways by modifying the highlighting inclusion and exclusion sets and retraversing the structure network. If the highlighting inclusion and exclusion sets are empty when the structure network is traversed, the scene is drawn as normal. If the user calls **PEXChangeNameSet** to add "antique" to the highlighting inclusion set, all of the "antique" objects are highlighted the next time the structure network is traversed. If the user adds "wooden" to the highlighting inclusion set, all "wooden" objects and all "antique" objects are highlighted. If "wooden" is added to the highlighting exclusion set instead, all objects that are "antique" but not "wooden" are highlighted.

The only attribute in a name set is a list of names. For performance reasons, the set of available names can be limited by an implementation. (See **PEXGetImpDepConstants**.) PEXlib includes functions to create, copy, free, query, and modify name sets.

PEXChangeNameSet - Change Name Set

Synopsis

void PEXChangeNameSet(Display **display*, PEXNameSet *nameset*, int *action*, unsigned long *count*, PEXName **names*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
nameset	The resource identifier of the name set.
action	Type of change to be made (**PEXNSAdd**, **PEXNSRemove**, **PEXNSReplace**).
count	The number of names.
names	An array of names.

Returns

None

Description

This function modifies the specified name set resource. The list of names is added to the name set if the specified action is **PEXNSAdd**, removed from the name set if the specified action is **PEXNSRemove**, or used to replace the names currently in the name set if the specified action is **PEXNSReplace**. If requested to remove a name that does not exist, it is silently ignored.

Data Structures

```
typedef XID              PEXNameSet;
typedef unsigned long    PEXName;
```

Errors

BadPEXNameSet	The specified name set resource identifier is invalid.
BadValue	The specified value for action parameter is invalid.

See Also

PEXCreateNameSet, **PEXGetNameSet**

PEXCopyNameSet - Copy Name Set

Synopsis

void PEXCopyNameSet(Display **display*, PEXNameSet *src_nameset*, PEXNameSet *dst_nameset*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
src_nameset	The resource identifier of the source name set.
dst_nameset	The resource identifier of the destination name set.

Returns

None

Description

This function copies the contents of the source name set to the destination name set. Both must already exist as valid name set resources. All values already in the destination name set are overwritten.

Data Structures

```
typedef XID             PEXNameSet;
```

Errors

BadPEXNameSet The specified name set resource identifier is invalid.

See Also

PEXCreateNameSet

PEXCreateNameSet - Create Name Set

Synopsis

PEXNameSet PEXCreateNameSet(Display **display*)

Argument

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

Returns

The resource identifier of the newly-created name set.

Description

This function creates a name set resource and returns the resource identifier of the created name set. The returned identifier is used to refer to the created name set.

Data Structures

```
typedef XID              PEXNameSet;
```

Errors

BadAlloc The server failed to allocate the resource.

See Also

PEXFreeNameSet

PEXFreeNameSet - Free Name Set

Synopsis

void PEXFreeNameSet(Display **display*, PEXNameSet *nameset*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
nameset	The resource identifier of the name set.

Returns

None

Description

This function deletes the association between the name set resource identifier and the name set. The name set is freed when no other resource references it.

Errors

BadPEXNameSet The specified name set resource identifier is invalid.

See Also

PEXCreateNameSet

PEXGetNameSet - Get Name Set

Synopsis

Status PEXGetNameSet(Display **display*, PEXNameSet *nameset*, unsigned long **count_return*,
PEXName ***names_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
nameset	The resource identifier of the name set to be queried.
count_return	Returns the number of names.
names_return	Returns an array of names.

Returns

Zero if unsuccessful; non-zero otherwise.

Description

This function returns the names in the specified name set.

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

Data Structures

```
typedef XID             PEXNameSet;
typedef unsigned long   PEXName;
```

Errors

BadPEXNameSet The specified name set resource identifier is invalid.

See Also

PEXCreateNameSet, **PEXChangeNameSet**

Chapter 9

Spatial Searches

The functions described in this chapter allow for spatial searches of structure networks. The portion of the structure network to be searched is specified, and a set of filters much like the invisibility and highlight filters are specified to limit the set of eligible primitives. A search distance is also specified. The first output primitive that satisfies the filter criteria and is within the search distance is returned.

9.1. Search Context

A search context is a PEX resource that contains all the search criteria to be used when spatially searching a structure network. Two of the attributes that specify search criteria contain name set resource identifiers. If name sets are created, bound to a search context, then freed, the contents of the name sets will remain, since they are still referenced by the search context. However, if the search context is queried, the value **PEXAlreadyFreed** will be returned for any such freed name sets, since they no longer have valid resource identifiers by which they can be referenced.

A search context resource contains the following attributes:

Search position
Search distance
Search ceiling
Model clip flag
Start path
Normal list
Inverted list

The search position attribute specifies the search reference position in world coordinates.

The search distance attribute specifies a distance from the search reference position in world coordinates. The distance specifies a search aperture. Whether the aperture is considered to be a sphere or a cube is implementation-dependent. If the search distance is less than or equal to zero, a primitive must intersect the search position to be considered within the search aperture.

The search ceiling attribute defines the ceiling of the search operation. The search ceiling is an index into the list of element references contained in the search start path. Index one refers to the first path element in the list. Searching stops when the end of the structure specified by the search ceiling is reached. If the ceiling is one, the search effectively operates without a ceiling and terminates at the end of the first structure in the start path.

The model clip flag specifies whether modeling clipping must be performed during the search operations. If **True**, modeling clipping is performed using the modeling clipping attributes as they are encountered during the search traversal. If **False**, no model clipping is performed and model clipping attributes encountered during the traversal are effectively ignored.

The start path attribute defines the structure network path used as the starting point for the search. The search traversal begins at the element following the one indicated by the start path.

The normal list is a list of name set resource identifier pairs. Each pair of name sets is considered to be a filter to be used in the search traversal. A primitive's name set attribute must have at least one name in common with the inclusion name set of each filter, and must have no names in common with the exclusion name set of each filter, to be eligible for the search.

The inverted list is a list of name set resource identifier pairs. Each pair of name sets is considered to be a filter that is inverted and used in the search traversal. A primitive's name set attribute must have no names in common with the inclusion name set of each inverted filter, or must have at least one name in common with the exclusion name set of each inverted filter, to be eligible for the search.

The **PEXSearchNetwork** function is used to perform a search operation once all of the search criteria attributes have been established. To search, the structure network specified is conceptually traversed, starting at the element following the element specified by the start path. Each output primitive is transformed to world coordinates and compared against the search aperture to determine proximity. For text and annotation text primitives, only the transformed origin of the text string is used to determine proximity. If the transformed primitive's geometry places it within the search aperture, its name set attribute is checked against the normal list and against the inverted list to determine eligibility.

The search is successful if an output primitive is found which satisfies the search filter criteria and is withing the search aperture. The path to the first such primitive found is returned, and the start path attribute of the search context is set to the found path. This allows the next search operation on the same search context to continue from just after the found primitive.

If no such primitive is found before the end of the structure indicated by the search ceiling is reached, the search is unsuccessful, NULL is returned as the found path, and the start path attribute is not changed.

PEXChangeSearchContext - Change Search Context

Synopsis

void PEXChangeSearchContext(Display **display*, PEXSearchContext *context*, unsigned long *value_mask*, PEXSCAttributes **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
context	The resource identifier of the search context.
value_mask	A mask indicating the search context attributes to be changed.
values	A pointer to new values for the specified search context attributes.

Returns

None

Description

This function modifies the attributes of the specified search context resource. The value mask indicates which values are specified. The value mask is constructed by or'ing together the following constants:

PEXSCPosition
PEXSCDistance
PEXSCCeiling
PEXSCModelClipFlag
PEXSCStartPath
PEXSCNormalList
PEXSCInvertedList

Data Structures

```
typedef XID             PEXSearchContext;

typedef struct {
    PEXCoord            position;
    float               distance;
    unsigned short      ceiling;
    Bool                model_clip_flag;
    PEXStructurePath    start_path;
    PEXListOfNameSetPair normal;
    PEXListOfNameSetPair inverted;
} PEXSCAttributes;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;
```

```
typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;

typedef XID             PEXStructure;

typedef struct {
    unsigned short      count;                  /* number of pairs */
    PEXNameSetPair      *pairs;
} PEXListOfNameSetPair;

typedef struct {
    PEXNameSet          inclusion;
    PEXNameSet          exclusion;
} PEXNameSetPair;

typedef XID             PEXNameSet;
```

Errors

BadPEXNameSet The specified name set resource identifier is invalid.

BadPEXPath The specified path is invalid.

BadPEXSearchContext
The specified search 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

PEXCreateSearchContext, **PEXGetSearchContext**

PEXCopySearchContext - Copy Search Context

Synopsis

void PEXCopySearchContext(Display **display*, unsigned long *value_mask*, PEXSearchContext *src_context*, PEXSearchContext *dst_context*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
value_mask	A mask specifying which attributes are to be copied.
src_context	The resource identifier of the source search context.
dst_context	The resource identifier of the destination search context.

Returns

None

Description

This function copies attributes from the source search context to the destination search context resource. Both must already exist as valid search context resources. Attributes indicated by the value mask are copied and the remainder of the attributes are left unchanged. The value mask is constructed by or'ing together the following constants:

PEXSCPosition
PEXSCDistance
PEXSCCeiling
PEXSCModelClipFlag
PEXSCStartPath
PEXSCNormalList
PEXSCInvertedList

Data Structures

```
typedef XID             PEXSearchContext;
```

Errors

BadPEXSearchContext
The specified search context resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreateSearchContext

PEXCreateSearchContext - Create Search Context

Synopsis

PEXSearchContext PEXCreateSearchContext(Display **display*, unsigned long *value_mask*, PEXSCAttributes **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
value_mask	A mask indicating the attributes specified.
values	A pointer to values used to override default values in the new search context resource.

Returns

The resource identifier of the newly-created search context resource.

Description

This function creates a search context and returns the resource identifier of the created search context resource. The value mask indicates which values are specified to override the default values. The value mask is constructed by or'ing together the following constants:

PEXSCPosition
PEXSCDistance
PEXSCCeiling
PEXSCModelClipFlag
PEXSCStartPath
PEXSCNormalList
PEXSCInvertedList

Data Structures

```
typedef XID             PEXSearchContext;

typedef struct {
    PEXCoord            position;
    float               distance;
    unsigned short      ceiling;
    Bool                model_clip_flag;
    PEXStructurePath    start_path;
    PEXListOfNameSetPair normal;
    PEXListOfNameSetPair inverted;
} PEXSCAttributes;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;
```

```
typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;

typedef XID             PEXStructure;

typedef struct {
    unsigned short      count;                  /* number of pairs */
    PEXNameSetPair      *pairs;
} PEXListOfNameSetPair;

typedef struct {
    PEXNameSet          inclusion;
    PEXNameSet          exclusion;
} PEXNameSetPair;

typedef XID             PEXNameSet;
```

Errors

BadAlloc	The server failed to allocate the resource.
BadPEXNameSet	The specified name set resource identifier is invalid.
BadPEXPath	The specified path is invalid.
BadValue	A specified value is out of range, or an invalid bit is set in the value mask.

See Also

PEXFreeSearchContext

PEXFreeSCAttributes - Free Storage Returned by PEXGetSearchContext

Synopsis

void PEXFreeSCAttributes(PEXSCAttributes **values*)

Arguments

values A pointer to the search context attribute values.

Returns

None

Description

This function deallocates memory returned by **PEXGetSearchContext**.

Errors

None

See Also

PEXGetSearchContext

PEXFreeSearchContext - Free Search Context

Synopsis

void PEXFreeSearchContext(Display **display*, PEXSearchContext *context*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
context	The resource identifier of the search context.

Returns

None

Description

This function deletes the search context resource and frees memory associated with it.

Errors

BadPEXSearchContext

The specified search context resource identifier is invalid.

See Also

PEXCreateSearchContext

PEXGetSearchContext - Get Search Context Attributes

Synopsis

PEXSCAttributes *PEXGetSearchContext(Display **display*, PEXSearchContext *context*, unsigned long *value_mask*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
context	The resource identifier of the search context to be queried.
value_mask	A mask indicating which attributes to return.

Returns

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

Description

This function returns the requested attribute values of the specified search context resource. The value mask indicates which attributes are to be returned. The value mask is constructed by or'ing together the following constants:

PEXSCPosition
PEXSCDistance
PEXSCCeiling
PEXSCModelClipFlag
PEXSCStartPath
PEXSCNormalList
PEXSCInvertedList

PEXlib allocates memory for the returned search context attribute values. **PEXFreeSCAttributes** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXSearchContext;

typedef struct {
    PEXCoord            position;
    float               distance;
    unsigned short      ceiling;
    Bool                model_clip_flag;
    PEXStructurePath    start_path;
    PEXListOfNameSetPair normal;
    PEXListOfNameSetPair inverted;
} PEXSCAttributes;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;
```

```
typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;

typedef XID             PEXStructure;

typedef struct {
    unsigned short      count;                  /* number of pairs */
    PEXNameSetPair      *pairs;
} PEXListOfNameSetPair;

typedef struct {
    PEXNameSet          inclusion;
    PEXNameSet          exclusion;
} PEXNameSetPair;

typedef XID             PEXNameSet;
```

Errors

BadPEXSearchContext
The specified search context resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreateSearchContext, **PEXChangeSearchContext**, **PEXFreeSCAttributes**

PEXSearchNetwork - Search Network

Synopsis

Status PEXSearchNetwork(Display **display*, PEXSearchContext *context*, PEXStructurePath ***path_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
context	The resource identifier of the search context.
path_return	Returns a pointer to a structure network path identifying the first primitive found.

Returns

Zero if unsuccessful; non-zero otherwise.

Description

This function searches a structure network according to the specified search criteria. The path to the first primitive found that satisfies the search criteria is returned. If no primitive is found that satisfies the criteria, a null pointer is returned.

After the search has been completed, the start path attribute of the specified search context will be set to the path returned, if a primitive was found.

PEXlib allocates memory for the returned structure path. **PEXFreeStructurePaths** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXSearchContext;

typedef struct {
    unsigned long       count;              /* number of elements */
    PEXElementRef       *elements;
} PEXStructurePath;

typedef struct {
    PEXStructure        structure;
    unsigned long       offset;
} PEXElementRef;

typedef XID             PEXStructure;
```

Errors

BadPEXPath The specified path is invalid.

BadPEXSearchContext
The specified search context resource identifier is invalid.

See Also

PEXCreateSearchContext, **PEXChangeSearchContext**, **PEXFreeStructurePaths**

Chapter 10

PHIGS Workstations

The PHIGS workstation resource encapsulates the PEX functionality to support the PHIGS abstraction of a "workstation". The PHIGS definition of a workstation states that a workstation consists of zero or one display space. In X, many display spaces (windows) are possible in a single workstation display. The PHIGS workstation resource ties together a single X drawable, a set of lookup tables, highlighting and invisibility name sets, and all of the state needed to make it perform as a PHIGS user expects a workstation should perform.

Each PHIGS workstation resource contains the following attributes defined at the time it is created:

target drawable
color approximation table
color table
depth cue table
edge bundle
interior bundle
marker bundle
light table
line bundle
pattern table
text bundle
text font table
highlight inclusion
highlight exclusion
invisibility inclusion
invisibility exclusion

These attributes can be inquired but not modified during the lifetime of the PHIGS workstation. These attributes function in a manner identical to their counterparts in a renderer resource. Note that there is no view table as in the renderer. The workstation has a built-in view table that can only be accessed through the defined workstation functions. However, this view table is functionally the same as that used by the renderer.

A workstation also contains the following attributes that can be modified or inquired with various functions:

posted structure networks
requested NPC subvolume
current NPC subvolume
requested viewport
current viewport
requested HLHSR mode
current HLHSR mode
defined views
display update mode
requested buffer mode
current buffer mode

The posted structure networks attribute maintains a list of structure resource identifiers, together with a priority value for each. The priority values are relative, and are specified at the time the structure network is added to the list (posted). Whenever a structure traversal is requested, each of the structure networks in this list is traversed. If multiple structures that would overlap on the screen are posted, the structure network with the higher priority is guaranteed to be visible. If two structures have the same priority and would overlap on the screen, the structure posted later has the higher priority.

The **PEXPostStructure** function is called to post structure networks to a PHIGS workstation resource, and the **PEXUnpostStructure** function is called to unpost a structure network. The **PEXUnpostAllStructures** function is called to reset the posted structure network list to the null list. Finally, given a structure resource id, the **PEXGetWorkstationPostings** function returns a list of all PHIGS workstations to which the structure has been posted.

The PHIGS workstation resource supports the concepts of requested changes and current values. The user can change the viewport, NPC subvolume, HLHSR mode and buffer mode attributes, but the changes are not actually applied until the next structure traversal occurs or the user calls the **PEXExecuteDeferredActions** function. Thus, there are pairs of attributes for the NPC subvolume, viewport, HLHSR mode and buffer mode: the requested value and the current value.

The list of defined views is similar to a view table. However, a PHIGS workstation must also be able to separate requested views from current views. Views also have an input priority value associated with them in PHIGS workstation resources. Because of these differences in semantics, the PHIGS workstation resource was designed to have a built-in view table rather than an associated view lookup table.

When doing a picking or mapping operation, it is necessary to map device coordinates through the inverse of the viewing/projection transform to determine the primitive that was selected. The reason for the view priority is that several views can produce a projection onto the same area of the screen. Thus, specifying a device coordinate point for the picking operation can be ambiguous. If there is more than one view that projects onto the specified device coordinate point, it is not clear which set of viewing parameters to use to compute the inverse transform. Adding a priority to each of the views removes the ambiguity.

View priorities are adjusted with the **PEXSetWorkstationViewPriority** function. The **PEXMapDCToWC** function is provided as an aid for mapping device coordinate points back into world coordinates. The function **PEXMapWCToDC** aids in mapping world coordinates to device coordinates. Views are modified with the **PEXSetWorkstationViewRep** function, but are not actually applied until the next structure traversal occurs or the user calls the **PEXExecuteDeferredActions** function.

The display update mode indicates how the PHIGS workstation should attempt to visualize changes that occur. The defined types include **PEXVisualizeEach**, **PEXVisualizeEasy**, **PEXVisualizeNone**, **PEXVisualizeWhenever**, and **PEXSimulateSome**. A type of **PEXVisualizeEach** means that each change is displayed as it occurs, even doing a retraversal of the posted structure network list if necessary. A type of **PEXVisualizeEasy** means that those changes which can be represented without doing a retraversal, or even those that require a retraversal if traversal is very fast, are displayed as they occur. A type of **PEXVisualizeNone** means that changes are applied to the structure, but the structure is not traversed until the application program explicitly requests it. A type of **PEXVisualizeWhenever** means that traversal of the structure is done when it is convenient to do so. A type of **PEXSimulateSome** means that the easy or fast changes are immediately displayed and others are simulated by such means as drawing a deleted primitive in the background color.

A PHIGS workstation contains the following attributes to maintain the state of the PHIGS workstation resource:

visual state
display surface
view update
workstation update
HLHSR update
buffer update
number of priorities

These attributes can be inquired., but are not modifiable by users; they are modified as side effects of various requests.

The visual state attribute maintains a value that indicates whether the image is correct, whether some parts of the image have been changed but simulated on the display, or whether there are deferred actions that have yet to be visualized.

The display surface indicates whether the PHIGS workstation has rendered anything on the associated drawable.

The view update, workstation update, HLHSR update and buffer update attributes specify whether there are any requested changes to the list of views, the NPC subvolume or viewport, the HLHSR mode or the buffer mode respectively.

A PHIGS workstation's posted structure networks can be retraversed explicitly by calling **PEX-RedrawAllStructures**. The **PEXUpdateWorkstation** function can be called to perform a traversal if the visual state is not correct. The **PEXGetWorkstationDynamics** function is called to obtain information about how changes to various attributes are visualized.

Structure traversal using a PHIGS workstation resource has side effects beyond those of a renderer. When a traversal is performed using a PHIGS workstation, the drawable is cleared to the color stored in entry zero of the associated color lookup table. Furthermore, all pending changes to the PHIGS workstation attributes are made current as a traversal begins. Finally, the displayed image state is set to correct at the end of the traversal.

Pick device descriptors are also a part of the workstation resource. They are defined in *Chapter 11: PHIGS Workstation Picking*.

If the drawable associated with a workstation resource is destroyed, the workstation resource is implicitly freed. The next access to the workstation will generate a **BadPEXWorkstation** error.

Lookup tables and name set resource can be bound to a workstation resource and then freed. The contents of these resources will remain, since they are still being referenced by the workstation. However, when a workstation's attributes are inquired, the value **PEXAlreadyFreed** will be returned for those resources that have been freed and thus no longer have a valid resource identifier by which they can be referenced.

PEXCreateWorkstation - Create Workstation

Synopsis

PEXWorkstation PEXCreateWorkstation(Display **display*, Drawable *drawable*, PEXLookupTable *line_bundle*, PEXLookupTable *marker_bundle*, PEXLookupTable *text_bundle*, PEXLookupTable *interior_bundle*, PEXLookupTable *edge_bundle*, PEXLookupTable *color_table*, PEXLookupTable *pattern_table*, PEXLookupTable *font_table*, PEXLookupTable *depth_cue_table*, PEXLookupTable *light_table*, PEXLookupTable *color_approx_table*, PEXNameSet *highlight_incl*, PEXNameSet *highlight_excl*, PEXNameSet *invisibility_incl*, PEXNameSet *invisibility_excl*, int *buffer_mode*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
line_bundle	The resource identifier of the line bundle lookup table.
marker_bundle	The resource identifier of the marker bundle lookup table.
text_bundle	The resource identifier of the text bundle lookup table.
interior_bundle	The resource identifier of the interior bundle lookup table.
edge_bundle	The resource identifier of the edge bundle lookup table.
color_table	The resource identifier of the color lookup table.
pattern_table	The resource identifier of the pattern lookup table.
font_table	The resource identifier of the text font lookup table.
depth_cue_table	The resource identifier of the depth cue lookup table.
light_table	The resource identifier of the light lookup table.
color_approx_table	The resource identifier of the color approximation lookup table.
highlight_incl	The name set used as the highlight inclusion set.
highlight_excl	The name set used as the highlight exclusion set.
invisibility_incl	The name set used as the invisibility inclusion set.
invisibility_excl	The name set used as the invisibility exclusion set.
buffer_mode	The workstation buffering mode (**PEXSingleBuffer** or **PEXDoubleBuffer**).

Returns

The resource identifier of the newly-created workstation.

Description

This function creates a PHIGS workstation resource and returns its resource identifier. The drawable specified is associated with the newly-created workstation resource. The named tables and name sets are also bound to the workstation resource for use during rendering. A view table that supports current and requested view table entries is allocated for the workstation automatically at creation time. The requests **PEXSetWorkstationViewRep** and **PEXGetWorkstationViewRep** is used to modify and inquire the PHIGS workstation view table.

If the workstation is to operate in double-buffered mode and double-buffering is supported (see **PEXGetImpDepConstants**), an additional image buffer will be created for the drawable in an implementation-dependent fashion. If the specified drawable is a pixmap, no double-buffering will be performed.

Data Structures

```
typedef XID             PEXWorkstation;
typedef XID             PEXLookupTable;
typedef XID             PEXNameSet;
```

Errors

BadAlloc The server failed to allocate the resource, or the server failed to allocate resources needed for double-buffering.

BadDrawable The specified drawable resource identifier is invalid.

BadMatch The specified drawable is unsupported, or a specified lookup table resource was not created with a compatible drawable.

BadPEXLookupTable
The specified lookup table resource identifier is invalid.

BadPEXNameSet The specified name set resource identifier is invalid.

BadValue The specified buffer mode is invalid.

See Also

PEXFreeWorkstation, **PEXGetImpDepConstants**

PEXExecuteDeferredActions - Execute Deferred Workstation Actions

Synopsis

void PEXExecuteDeferredActions(Display **display*, PEXWorkstation *workstation*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.

Returns

None

Description

This function causes all deferred actions on the specified workstation to be executed. This will cause all requested attributes to be made current and the corresponding update attributes to be set to **PEX-NotPending**.

Data Structures

```
typedef XID              PEXWorkstation;
```

Errors

BadPEXWorkstation

The specified workstation resource identifier is invalid.

See Also

PEXRedrawAllStructures

PEXFreeWorkstation - Free Workstation

Synopsis

void PEXFreeWorkstation(Display **display*, PEXWorkstation *workstation*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.

Returns

None

Description

This function deletes the PHIGS workstation resource and frees memory associated with it.

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXCreateWorkstation

PEXFreeWorkstationAttributes - Free Storage Returned by PEXGetWorkstationAttributes

Synopsis

void PEXFreeWorkstationAttributes(PEXWorkstationAttributes **values*)

Arguments

values A pointer to the workstation attribute values.

Returns

None

Description

This function deallocates memory returned by **PEXGetWorkstationAttributes**.

Errors

None

See Also

PEXGetWorkstationAttributes

PEXGetWorkstationDynamics - Get Workstation Attribute Modification Dynamics

Synopsis

Status PEXGetWorkstationDynamics(Display **display*, Drawable *drawable*, PEXWorkstation-Dynamics **dynamics_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
drawable	The resource identifier of a drawable.
dynamics_return	Returns the dynamics of the specified workstation.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns the dynamics of the specified PHIGS workstation resource. Each entry in the returned list has one of the following values: **PEXIMM**, **PEXCBS** or **PEXIRG**. **PEXIMM** means that modification of the corresponding attribute will result in the correct image displayed immediately without regeneration. **PEXCBS** means that modification of the corresponding attribute will result in a simulation displayed immediately if the display update mode is **PEXSimulateSome.** **PEXIRG** means that a regeneration is required for the change to be visible.

Data Structures

```
typedef struct {
    unsigned char       view_rep;
    unsigned char       marker_bundle;
    unsigned char       text_bundle;
    unsigned char       line_bundle;
    unsigned char       interior_bundle;
    unsigned char       edge_bundle;
    unsigned char       color_table;
    unsigned char       pattern_table;
    unsigned char       wks_transform;
    unsigned char       highlight_filter;
    unsigned char       invisibility_filter;
    unsigned char       hlhsr_mode;
    unsigned char       structure_modify;
    unsigned char       post_structure;
    unsigned char       unpost_structure;
    unsigned char       delete_structure;
    unsigned char       reference_modify;
    unsigned char       buffer_modify;
    unsigned char       light_table;
    unsigned char       depth_cue_table;
    unsigned char       color_approx_table;
} PEXWorkstationDynamics;
```

Errors

BadDrawable	The specified drawable resource identifier is invalid.
BadMatch	The specified drawable is unsupported.

See Also

PEXGetWorkstationAttributes

PEXGetWorkstationAttributes - Get Workstation Attribute Values

Synopsis

PEXWorkstationAttributes *PEXGetWorkstationAttributes(Display *display, PEXWorkstation workstation, unsigned long *value_mask)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
value_mask	A mask indicating which workstation attribute values to return.

Returns

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

Description

This function returns the specified workstation's attributes. The value mask indicates which attributes are to be returned. **PEXSetPWAttributeMask** or **PEXSetPWAttributeMaskAll** should be used to setup the value mask.

PEXlib allocates memory for the returned workstation attribute values. **PEXFreeWorkstationAttributes** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    short               drawable_update;
    int                 visual_state;
    int                 drawable_surface;
    int                 view_update;
    PEXListOfView       defined_views;
    int                 wks_update;
    PEXNPCSubVolume     req_npc_subvolume;
    PEXNPCSubVolume     cur_npc_subvolume;
    PEXViewport         req_workstation_viewport;
    PEXViewport         cur_workstation_viewport;
    int                 hlhsr_update;
    PEXEnumTypeIndex    req_hlhsr_mode;
    PEXEnumTypeIndex    cur_hlhsr_mode;
    Drawable            drawable;
    PEXLookupTable      marker_bundle;
    PEXLookupTable      text_bundle;
    PEXLookupTable      line_bundle;
    PEXLookupTable      interior_bundle;
    PEXLookupTable      edge_bundle;
    PEXLookupTable      color_table;
    PEXLookupTable      depth_cue_table;
    PEXLookupTable      light_table;
    PEXLookupTable      color_approx_table;
    PEXLookupTable      pattern_table;
    PEXLookupTable      text_font_table;
    PEXNameSet          highlight_incl;
    PEXNameSet          highlight_excl;
```

```
    PEXNameSet          invisibility_incl;
    PEXNameSet          invisibility_excl;
    PEXListOfPostedStructure posted_structures;
    unsigned long       count_priorities;
    int                 buffer_update;
    int                 req_buffer_mode;
    int                 cur_buffer_mode;
} PEXWorkstationAttributes;

typedef struct {
    unsigned short      count;                  /* number of views */
    PEXTableIndex       *views;
} PEXListOfView;

typedef unsigned short  PEXTableIndex;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord      min;
    PEXDeviceCoord      max;
    PEXSwitch           use_drawable;
    unsigned char       reserved[3];
} PEXViewport;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef unsigned char   PEXSwitch;

typedef short           PEXEnumTypeIndex;
typedef XID             PEXLookupTable;
typedef XID             PEXNameSet;

typedef struct {
    unsigned long       count;                  /* number of posted structures*/
    PEXPostedStructure  *structures;
} PEXListOfPostedStructure;

typedef struct {
    PEXStructure        sid;
    float               priority;
```

```
} PEXPostedStructure;

typedef XID              PEXStructure;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXGetWorkstationDynamics, **PEXSetPWAttributeMask**, **PEXSetPWAttributeMaskAll**

PEXGetWorkstationPostings - Get Workstation Postings

Synopsis

Status PEXGetWorkstationPostings(Display **display*, PEXStructure *structure*, unsigned long **count_return*, PEXWorkstation ***postings_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
structure	The resource identifier of the structure.
count_return	Returns the number of workstation identifiers.
postings_return	Returns an array of workstation resource identifiers.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function returns a list of workstation resources to which the specified structure has been posted.

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

Data Structures

```
typedef XID            PEXStructure;
typedef XID            PEXWorkstation;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXPostStructure, **PEXUnpostStructure**, **PEXUnpostAllStructures**

PEXGetWorkstationViewRep - Get Workstation View Representation

Synopsis

Status PEXGetWorkstationViewRep(Display **display*, PEXWorkstation *workstation*, unsigned int *index*, int **update_return*, PEXViewRep **req_view_return*, PEXViewRep **cur_view_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
index	The view table index.
update_return	Returns the update status of the view table index.
req_view_return	Returns the requested value of the view table index.
cur_view_return	Returns the current value of the view table index.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

The function returns the update state, and the requested and current values for the specified view index of the specified workstation. The update will be **PEXPending** if a view change has been requested but not established; otherwise it will be **PEXNotPending**. If the specified view index is not defined, an error will be generated and the contents of the reply parameters will be undefined.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    PEXTableIndex       index;
    unsigned short      reserved;
    PEXViewEntry        view;
} PEXViewRep;

typedef unsigned short  PEXTableIndex;

typedef struct {
    unsigned short      clip_flags;
    unsigned short      reserved;
    PEXNPCSubVolume     clip_limits;
    PEXMatrix           orientation;
    PEXMatrix           mapping;
} PEXViewEntry;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
```

```
        float               z;
    } PEXCoord;

    typedef float           PEXMatrix[4][4];
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue The specified view table entry is not defined.

See Also

PEXSetWorkstationViewRep

PEXMapDCToWC - Map Device Coordinate Points to World Coordinate Points

Synopsis

Status PEXMapDCToWC(Display **display*, PEXWorkstation *workstation*, unsigned long *dc_count*, PEXDeviceCoord **dc_points*, unsigned int **view_index_return*, unsigned long **wc_count_return*, PEXCoord ***wc_points_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
dc_count	The number of device coordinate points.
dc_points	An array of device coordinates.
view_index_return	Returns the view index of the view containing most or all of the points.
wc_count_return	Returns the number of world coordinate points.
wc_points_return	Returns an array of world coordinate points.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function maps the device coordinate points to world coordinate points using the specified workstation. Each view in the workstation's current view table, which has an inverse, is checked to see if it contains all the specified device coordinate points. (If the view transform has no inverse, it is not considered.) The index of the view with the highest view transformation input priority that contains all of the points is returned. If no view contains all the points, the index of the view containing the most points is returned. The points are transformed to world coordinates by passing them through the inverse of the view transform associated with the view index. Points that are clipped (outside the viewport) will not be transformed and returned, so the number of points returned may be less than the number sent.

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

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXMapWCToDC, **PEXSetWorkstationViewPriority**

PEXMapWCToDC - Map World Coordinate Points to Device Coordinate Points

Synopsis

Status PEXMapWCToDC(Display **display*, PEXWorkstation *workstation*, unsigned long *wc_count*, PEXCoord **wc_points*, unsigned int *view_index*, unsigned long **dc_count_return*, PEXDeviceCoord ***dc_points_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
wc_count	The number of world coordinate points.
wc_points	An array of world coordinate points.
view_index	The view index to use in transforming the world coordinate points.
dc_count_return	Returns the number of device coordinate points.
dc_points_return	Returns an array of device coordinate points.

Returns

Zero if unsuccessful, non-zero otherwise.

Description

This function maps the world coordinate points to device coordinate points using the specified workstation and view index. The points are transformed to device coordinates by passing them through the view transform associated with the view index. Points that are clipped will not be returned, so the number of points returned may be less than the number sent.

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

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXMapDCToWC, **PEXSetWorkstationViewPriority**

PEXPostStructure - Post Structure to a Workstation

Synopsis

void PEXPostStructure(Display **display*, PEXWorkstation *workstation*, PEXStructure *structure*, double *priority*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
structure	The resource identifier of the structure.
priority	The priority of the newly-posted structure.

Returns

None

Description

This function adds the specified structure to the list of posted structures in the specified workstation. The priority is used to indicate the priority of the new posted structure relative to the structures already posted. If multiple structures are posted for display to the same display space location, the higher priority structure will be displayed. If two structures have the same priority, the last posted structure is considered of higher priority.

Data Structures

```
typedef XID             PEXStructure;
typedef XID             PEXWorkstation;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXUnpostStructure, **PEXUnpostAllStructures**

PEXRedrawAllStructures - Redraw All Posted Structures

Synopsis

void PEXRedrawAllStructures(Display **display*, PEXWorkstation *workstation*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.

Returns

None

Description

This function redraws all of the posted structures for the workstation. First, if the workstation's display surface is **PEXNotEmpty**, its drawable is cleared to the color stored in entry zero of the color table. Then, if any of its view, workstation, HLHSR or buffer update attributes is set to **PEXPending**, the requested values are made current and the update attributes are set to **PEXNotPending**. After this, all the posted structures are traversed and rendered (in priority order). Finally, the workstation's visual state is set to **PEXCorrect**.

Data Structures

```
typedef XID               PEXWorkstation;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXUpdateWorkstation, **PEXExecuteDeferredActions**

PEXRedrawClipRegion - Redraw Posted Structures Clipped to Clip Region

Synopsis

void PEXRedrawClipRegion(Display **display*, PEXWorkstation *workstation*, unsigned long *count*,
PEXDeviceRect **clip_rectangles*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
count	The number of clip rectangles.
clip_rectangles	An array of device clip rectangles.

Returns

None

Description

This function performs actions similar to **PEXRedrawAllStructures** except that the workstation state attributes are not modified or updated by this function and rendering will occur only in the region defined by the specified clip list. The color stored in entry zero of the color table is used to clear the region defined by the clip list. The clip list must consist of non-overlapping rectangles or the result will be undefined. If a z-buffering HLHSR algorithm is used, only the z-values corresponding to pixels in the clip region will be affected. All of the posted structures for the workstation are redrawn, but clipped to the clip list. Pending changes are not made current, nor is the visual state modified.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    short               xmin;
    short               ymin;
    short               xmax;
    short               ymax;
} PEXDeviceRect;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXRedrawAllStructures, **PEXExecuteDeferredActions**

PEXSetWorkstationBufferMode - Set Workstation Buffer Mode

Synopsis

void PEXSetWorkstationBufferMode(Display **display*, PEXWorkstation *workstation*, int *buffer_mode*)

Arguments

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

workstation The resource identifier of the workstation.

buffer_mode The workstation buffering mode (**PEXSingleBuffer** or **PEXDoubleBuffer**).

Returns

None

Description

This function sets the requested buffer mode of the specified workstation. If the workstation's display surface attribute is **PEXEmpty**, or if the dynamic modification for buffer mode is **PEXIMM**, the current buffer mode is set to the specified buffer mode and the buffer update is set to **PEXNotPending**; otherwise, the buffer update is set to **PEXPending** and the current buffer mode is not changed.

Buffer mode may be one of the following values: **PEXSingleBuffer** or **PEXDoubleBuffer**. An error will be generated if the buffer mode is **PEXDoubleBuffer** and the server cannot allocated the second image buffer.

Data Structures

```
typedef XID             PEXWorkstation;
```

Errors

BadAlloc The server failed to allocate resources needed for double-buffering.

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue The specified buffer mode is invalid.

See Also

PEXSetWorkstationDisplayUpdateMode, **PEXGetImpDepConstants**

PEXSetWorkstationDisplayUpdateMode - Set Workstation Display Update Mode

Synopsis

void PEXSetWorkstationDisplayUpdateMode(Display **display*, PEXWorkstation *workstation*, int *update_mode*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
update_mode	The workstation display update mode (**PEXVisualizeEach**, **PEXVisualizeEasy**, **PEXVisualizeNone**, **PEXSimulateSome**, **PEXVisualizeWhenever**).

Returns

None

Description

This function sets the display update attribute of the specified workstation. This attribute defines how changes to the display surface will be visualized. The supported values for display update mode are inquirable via **PEXGetEnumTypeInfo**.

If double-buffering is enabled, the display update mode affects which buffer is rendered into during traversal. If the display update mode is **PEXVisualizeEach**, **PEXVisualizeWhenever** or **PEXVisualizeNone**, output primitives are rendered into the back (undisplayed) buffer while the structure network is being traversed. When the traversal is complete, the front and back buffers are swapped, so the rendered image is displayed. If the display update mode is **PEXVisualizeEasy** or **PEXSimulateSome**, output primitives are always rendered into the front (displayed) buffer.

Data Structures

```
typedef XID             PEXWorkstation;
```

Errors

BadPEXWorkstation
: The specified workstation resource identifier is invalid.

BadValue
: The specified display update mode is invalid.

See Also

PEXSetWorkstationBufferMode, **PEXGetEnumTypeInfo**

PEXSetWorkstationHLHSRMode - Set Workstation HLHSR Mode

Synopsis

void PEXSetWorkstationHLHSRMode(Display **display*, PEXWorkstation *workstation*, int *hlhsr_mode*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
hlhsr_mode	The workstation HLHSR mode (**PEXHLHSROff**, **PEXHLHSRZBuffer**, **PEXHLHSRPainters**, **PEXHLHSRScanline**, **PEXHLHSRHiddenLineOnly**, **PEXHLHSRZBufferID**).

Returns

None

Description

This function sets the requested HLHSR mode of the specified workstation. If the workstation's display surface attribute is **PEXEmpty**, or if the dynamic modification for HLHSR mode is **PEXIMM**, the current HLHSR mode is set to the specified HLHSR mode and the HLHSR update is set to **PEXNotPending**; otherwise, the HLHSR update is set to **PEXPending** and the current HLHSR mode is not changed.

Data Structures

```
typedef XID              PEXWorkstation;
```

Errors

BadAlloc	The server failed to allocate resources needed for HLHSR.
BadPEXWorkstation	The specified workstation resource identifier is invalid.
BadValue	The specified HLHSR mode is invalid.

See Also

PEXSetWorkstationViewport - Set Workstation Viewport

Synopsis

void PEXSetWorkstationViewport(Display **display*, PEXWorkstation *workstation*, PEXViewport **viewport*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
viewport	The workstation viewport.

Returns

None

Description

This function sets the requested workstation viewport of the specified workstation. If the dynamic modification for workstation viewport is **PEXIMM**, the current workstation viewport is set to the specified viewport and the workstation update is set to **PEXNotPending**; otherwise, the workstation update is set to **PEXPending** and the current workstation viewport is not changed.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    PEXDeviceCoord      min;
    PEXDeviceCoord      max;
    PEXSwitch           use_drawable;
    unsigned char       reserved[3];
} PEXViewport;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef unsigned char   PEXSwitch;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue The specified value for use_drawable is invalid.

See Also

PEXSetWorkstationViewPriority - Set Workstation View Priority

Synopsis

void PEXSetWorkstationViewPriority(Display **display*, PEXWorkstation *workstation*, unsigned int *index1*, unsigned int *index2*, int *priority*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
index1	The first view table entry index.
index2	The second view table entry index.
priority	A relative priority of index 1 with respect to index 2 (**PEXHigher** or **PEXLower**).

Returns

None

Description

This function sets the relative priorities of entries in the workstation's current view table. The priority of the first view table entry with respect to the second view table entry is set to the next higher or lower priority as indicated by the specified priority. These priorities are used to determine the order in which view table entries are tested when selecting the inverse viewing transformation to use for transforming from device coordinates to world coordinates.

Data Structures

```
typedef XID              PEXWorkstation;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue The specified value for priority is invalid, or the specified table entry is not defined.

See Also

PEXSetWorkstationViewRep - Set Workstation View Representation

Synopsis

void PEXSetWorkstationViewRep(Display **display*, PEXWorkstation *workstation*, unsigned int *view_index*, PEXViewEntry **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
view_index	The view index.
values	A pointer to the view representation values.

Returns

None

Description

This function sets the specified view table entry of the requested view table in the specified workstation. If the dynamic modification for the view table is **PEXIMM**, the current view table entry is set to the specified view representation and the view update is set to **PEXNotPending**; otherwise, the view update is set to **PEXPending** and the current view table is not changed.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    unsigned short      clip_flags;
    unsigned short      reserved;
    PEXNPCSubVolume     clip_limits;
    PEXMatrix           orientation;
    PEXMatrix           mapping;
} PEXViewEntry;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef float           PEXMatrix[4][4];
```

Errors

BadAlloc	The view table is full.
BadPEXWorkstation	The specified workstation resource identifier is invalid.

See Also

PEXSetWorkstationWindow - Set Workstation Window

Synopsis

void PEXSetWorkstationWindow(Display **display*, PEXWorkstation *workstation*, PEXNPCSubVolume **workstation_window*)

Arguments

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

workstation The resource identifier of the workstation.

workstation_window
The workstation window.

Returns

None

Description

This function set the requested NPC subvolume of the specified workstation. If the dynamic modification for the workstation window is **PEXIMM**, the current workstation window is set to the specified NPC subvolume and the workstation update is set to **PEXNotPending**; otherwise, the workstation update is set to **PEXPending** and the current workstation window is not changed.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXUnpostAllStructures - Unpost All Structures from Workstation

Synopsis

void PEXUnpostAllStructures(Display **display*, PEXWorkstation *workstation*)

Arguments

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

workstation The resource identifier of the workstation.

Returns

None

Description

This function removes all structures from the specified workstation's posted structure list.

Data Structures

```
typedef XID              PEXWorkstation;
```

Errors

BadPEXWorkstation

The specified workstation resource identifier is invalid.

See Also

PEXUnpostStructure

PEXUnpostStructure - Unpost Structure from Workstation

Synopsis

void PEXUnpostStructure(Display **display*, PEXWorkstation *workstation*, PEXStructure *structure*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
structure	The resource identifier of the structure.

Returns

None

Description

This function removes the specified structure from the workstation's posted structure list. If the structure is not found in the list, an error is generated.

Data Structures

```
typedef XID             PEXWorkstation;
typedef XID             PEXStructure;
```

Errors

BadPEXStructure The specified structure resource identifier is invalid.

BadPEXWorkstation
The specified workstation resource identifier is invalid.

See Also

PEXUnpostAllStructures

PEXUpdateWorkstation - Update Workstation

Synopsis

void PEXUpdateWorkstation(Display **display*, PEXWorkstation *workstation*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.

Returns

None

Description

This function will perform actions identical to **PEXRedrawAllStructures** on the specified workstation if the workstation's visual state is currently set to **PEXDeferred** or **PEXSimulated**.

Data Structures

```
typedef XID              PEXWorkstation;
```

Errors

BadPEXWorkstation

The specified workstation resource identifier is invalid.

See Also

PEXRedrawAllStructures

PEXSetPWAttributeMask - Macro to Setup Workstation Attributes Value Mask

Synopsis

PEXSetPWAttributeMask(*mask*, *attr*)

Arguments

mask	The address of the value mask - an array of two unsigned long.
attr	A single workstation attribute bitmask constant.

Description

This is a utility macro to aid in setting up the bitmask for the workstation attributes.

The following constants must be used:

PEXPWDisplayUpdate
PEXPWVisualState
PEXPWDisplaySurface
PEXPWViewUpdate
PEXPWDefinedViews
PEXPWWorkstationUpdate
PEXPWReqNPCSubVolume
PEXPWCurNPCSubVolume
PEXPWReqViewport
PEXPWCurViewport
PEXPWHLHSRUpdate
PEXPWReqHLHSRMode
PEXPWCurHLHSRMode
PEXPWDrawable
PEXPWMarkerBundle
PEXPWTextBundle
PEXPWLineBundle
PEXPWInteriorBundle
PEXPWEdgeBundle
PEXPWColorTable
PEXPWDepthCueTable
PEXPWLightTable
PEXPWColorApproxTable
PEXPWPatternTable
PEXPWTextFontTable
PEXPWHighlightIncl
PEXPWHighlightExcl
PEXPWInvisibilityIncl
PEXPWInvisibilityExcl
PEXPWPostedStructures
PEXPWNumPriorities
PEXPWBufferUpdate
PEXPWReqBufferMode
PEXPWCurBufferMode

Note that this macro does multiple evaluations of the value for attr.

Errors

None

See Also

PEXGetWorkstationAttributes

PEXSetPWAttributeMaskAll - Macro to Set All Workstation Attributes in Value Mask

Synopsis

PEXSetPWAttributeMaskAll(*mask*)

Arguments

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 workstation attributes. This macro will set all valid bits in the mask.

Errors

None

See Also

PEXGetWorkstationAttributes

Chapter 11

PHIGS Workstation Picking

The PHIGS workstation resource encapsulates the PEX functionality to support the PHIGS concept of a workstation. The following functionality encapsulates the picking support for the PHIGS workstation. There are two parts to the PHIGS workstation picking. A pick device is actually a part of the PHIGS workstation resource, but the support for inquiring and setting the pick device attributes is documented here. A pick measure is a separate resource that works with the PHIGS workstation resource to perform the actual pick operation.

11.1. Workstation Pick Devices

Each PHIGS workstation resource maintains a list of pick device descriptors. Each entry in the list maintains state values for a particular type of pick device, such as a mouse or a tablet. Together, the entries in the pick device descriptor list maintain state values for all the pick devices that are supported by the workstation resource. This list of values can be set or inquired for each of the supported pick devices.

Two of the pick device descriptor attributes are name set resources. If a name set is created, bound to a pick device descriptor, and the freed, the contents of the name set will remain, since it is still being referenced by the pick device descriptor. However, when a pick device descriptor is inquired, the value **PEXAlreadyFreed** will be returned for the name set identifier, since it no longer has a valid resource identifier by which it can be referenced.

The pick device descriptor attributes, in order, are:

pick status
pick path
pick path order
pick inclusion
pick exclusion
pick data record
prompt and echo type
echo volume
echo switch

The pick status attribute contains the initial pick status that will be bound to a pick measure resource.

The pick path attribute contains the initial pick path that will be bound to a pick measure resource.

The pick path order attribute specifies the order in which elements of the picked path are to be written into the pick measure resource. If the order is **PEXTopFirst**, elements of the pick measure's picked path attribute will be listed in the order they would have been encountered during a traversal. If the order is **PEXBottomFirst**, elements of the pick measure's picked path attribute will be listed in the order opposite from the order they would have been encountered during a traversal.

The pick inclusion attribute specifies the resource identifier of the name set that is to be used as the pick inclusion filter during pick operations.

The pick exclusion attribute specifies the resource identifier of the name set that is to be used as the pick exclusion filter during pick operations.

The pick data record attribute contains a pick-device-dependent data record used to initialize a pick measure resource when it is created.

The prompt and echo type attribute contains a value that specifies the prompt and echo type to be used during pick operations. The supported types are inquirable via **PEXGetEnumTypeInfo**.

The echo volume attribute specifies where prompting and echoing is to occur. The default is that the echo volume will be defined to be the size of the drawable bound to the workstation resource at the time it was created.

The echo switch attribute specifies the initial echo state for a pick measure.

PEXFreePDAttributes - Free Storage Returned by PEXGetPickDevice

Synopsis

void PEXFreePDAttributes(PEXPDAttributes **values*)

Arguments

values A pointer to the pick device attribute values.

Returns

None

Description

This function deallocates memory returned by **PEXGetPickDevice**.

Errors

None

See Also

PEXGetPickDevice

PEXGetPickDevice - Get Pick Device Attributes

Synopsis

PEXPDAttributes *PEXGetPickDevice(Display **display*, PEXWorkstation *workstation*, int *pick_device_type*, unsigned long *value_mask*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
value_mask	A mask indicating which attributes to return.

Returns

A pointer to the pick device attribute values; a null pointer if unsuccessful.

Description

This function returns the attribute values of a pick descriptor for the PHIGS workstation resource specified. The descriptor returned will be the currently-defined descriptor for the pick device of the type specified. Supported pick device types are inquirable via **PEXGetEnumTypeInfo**. The value mask indicates which attributes are to be returned. The value mask is constructed by or'ing together the following constants:

PEXPDPickStatus
PEXPDPickPath
PEXPDPickPathOrder
PEXPDPickIncl
PEXPDPickExcl
PEXPDPickDataRec
PEXPDPromptEchoType
PEXPDEchoVolume
PEXPDEchoSwitch

PEXlib allocates memory for the return value. **PEXFreePDAttributes** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    unsigned short      status;
    PEXPickPath         path;
    int                 path_order;
    PEXNameSet          inclusion;
    PEXNameSet          exclusion;
    PEXPickRecord       pick_record;
    PEXEnumTypeIndex    prompt_echo_type;
    PEXViewport         echo_volume;
    int                 echo_switch;
} PEXPDAttributes;

typedef struct {
```

```
    unsigned long       count;                  /* number of elements */
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
} PEXPickElementRef;

typedef XID             PEXStructure;

typedef XID             PEXNameSet;

typedef union {
    PEXPDNPCHitVolume       volume;
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;

typedef short           PEXEnumTypeIndex;

typedef struct {
    PEXDeviceCoord      min;
    PEXDeviceCoord      max;
    PEXSwitch           use_drawable;
```

```
    unsigned char       reserved[3];
} PEXViewport;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef unsigned char   PEXSwitch;
```

Errors

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue The specified pick device type is invalid, or an invalid bit set in the value mask.

See Also

PEXChangePickDevice, **PEXGetEnumTypeInfo**

PEXChangePickDevice - Change Pick Device Attributes

Synopsis

void PEXChangePickDevice(Display **display*, PEXWorkstation *workstation*, int *pick_device_type*, unsigned long *value_mask*, PEXPDAttributes **values*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).
value_mask	A mask indicating which attributes to return.
values	A pointer to the pick device attribute values.

Returns

None

Description

This function will modify the attributes of a pick descriptor for the PHIGS workstation resource specified. The descriptor to be modified will be the currently-defined descriptor for the pick device of the type specified. Supported pick device types are inquirable via **PEXGetEnumTypeInfo**. The value mask indicates which attributes are to be changed. The value mask is constructed by or'ing together the following constants:

PEXPDPickStatus
PEXPDPickPath
PEXPDPickPathOrder
PEXPDPickIncl
PEXPDPickExcl
PEXPDPickDataRec
PEXPDPromptEchoType
PEXPDEchoVolume
PEXPDEchoSwitch

Data Structures

```
typedef XID             PEXWorkstation;

typedef struct {
    unsigned short      status;
    PEXPickPath         path;
    int                 path_order;
    PEXNameSet          inclusion;
    PEXNameSet          exclusion;
    PEXPickRecord       pick_record;
    PEXEnumTypeIndex    prompt_echo_type;
    PEXViewport         echo_volume;
    int                 echo_switch;
} PEXPDAttributes;

typedef struct {
    unsigned long       count;                  /* number of elements */
```

```
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
} PEXPickElementRef;

typedef XID             PEXStructure;

typedef XID             PEXNameSet;

typedef union {
    PEXPDNPCHitVolume       volume;
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;

typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;

typedef short           PEXEnumTypeIndex;

typedef struct {
    PEXDeviceCoord      min;
    PEXDeviceCoord      max;
    PEXSwitch           use_drawable;
    unsigned char       reserved[3];
```

```
} PEXViewport;

typedef struct {
    short               x;
    short               y;
    float               z;
} PEXDeviceCoord;

typedef unsigned char   PEXSwitch;
```

Errors

BadPEXNameSet The specified name set resource identifier is invalid.

BadPEXPath The specified path is invalid.

BadPEXWorkstation
The specified workstation resource identifier is invalid.

BadValue The specified pick device type is invalid, a specified value is invalid, or an invalid bit set in the value mask.

See Also

PEXGetPickDevice, **PEXGetEnumTypeInfo**

11.2. Workstation Pick Measures

A pick measure resource must be created to actually perform a pick operation using a PHIGS workstation. A pick device type is specified at the time a pick measure resource is created in order to provide the parameters for the pick operation. The pick measure accepts input in the form of input records which are defined for each type of pick device. When the pick measure is passed a valid input record, its attributes will be updated. Operation on a pick measure are potentially lengthy since a great number of structure elements may have to be processed in order to produce the pick results.

The pick measure resource attributes are:

status
pick path

The pick status attribute contains the result of the last update operation that was performed. It is set to **PEXPick** if a primitive was successfully picked, and **PEXNoPick** if the pick operation was unsuccessful.

The pick path attribute contains the path of the structure element that was picked as a result of the last update operation.

PEXCreatePickMeasure - Create Pick Measure

Synopsis

PEXPickMeasure PEXCreatePickMeasure(Display **display*, PEXWorkstation *workstation*, int *pick_device_type*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
workstation	The resource identifier of the workstation.
pick_device_type	The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).

Returns

The resource identifier of the newly-created pick measure.

Description

This function creates a pick measure resource of the type specified. The pick measure is initialized with the values contained in the appropriate pick device descriptor stored in the specified workstation. The supported pick device types are inquirable via **PEXGetEnumTypeInfo**.

Data Structures

```
typedef XID              PEXWorkstation;
typedef XID              PEXPickMeasure;
```

Errors

BadAlloc	The server failed to allocate the resource.
BadPEXWorkstation	The specified workstation resource identifier is invalid.
BadValue	The specified pick device type is invalid.

See Also

PEXFreePickMeasure, **PEXGetEnumTypeInfo**

PEXFreePickMeasure - Free Pick Measure

Synopsis

void PEXFreePickMeasure(Display **display*, PEXPickMeasure *pick_measure*)

Arguments

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

pick_measure The resource identifier of the pick measure.

Returns

None

Description

This function deletes the pick measure resource and frees memory associated with it.

Errors

BadPEXPickMeasure

The specified pick measure resource identifier is invalid.

See Also

PEXCreatePickMeasure

PEXFreePMAttributes - Free Storage Returned by PEXGetPickMeasure

Synopsis

void PEXFreePMAttributes(PEXPMAttributes **values*)

Arguments

values A pointer to the pick measure attribute values.

Returns

None

Description

This function deallocates memory returned by **PEXGetPickMeasure**.

Errors

None

See Also

PEXGetPickMeasure

PEXGetPickMeasure - Get Pick Measure Attributes

Synopsis

PEXPMAttributes *PEXGetPickMeasure(Display **display*, PEXPickMeasure *pick_measure*, unsigned long *value_mask*)

Arguments

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

pick_measure The resource identifier of the pick measure.

value_mask A mask indicating which attributes to return.

Returns

A pointer to the pick measure attribute values; a null pointer if unsuccessful.

Description

This function returns the specified pick measure attribute values. The value mask indicates which attributes are to be returned. The value mask is constructed by or'ing together the following constants:

PEXPMStatus
PEXPMPath

PEXlib allocates memory for the return value. **PEXFreePMAttributes** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXPickMeasure;

typedef struct {
    unsigned short      status;
    PEXPickPath         pick_path;
} PEXPMAttributes;

typedef struct {
    unsigned long       count;                  /* number of elements */
    PEXPickElementRef   *elements;
} PEXPickPath;

typedef struct {
    PEXStructure        sid;
    unsigned long       offset;
    unsigned long       pick_id;
} PEXPickElementRef;

typedef XID             PEXStructure;
```

Errors

BadPEXPickMeasure
The specified pick measure resource identifier is invalid.

BadValue An invalid bit is set in the value mask.

See Also

PEXCreatePickMeasure

PEXUpdatePickMeasure - Update Pick Measure

Synopsis

void PEXUpdatePickMeasure(Display **display*, PEXPickMeasure *pick_measure*, int *pick_device_type*, PEXPickRecord **pick_record*)

Arguments

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

pick_measure The resource identifier of the pick measure.

pick_device_type The pick device type (**PEXPickDeviceDCHitBox** or **PEXPickDeviceNPCHitVolume**).

pick_record A pointer to the pick data record.

Returns

None

Description

This function updates the specified pick measure resource. If the update causes a primitive to be picked, the pick measure's pick status will be set to **PEXPick** and the pick path attribute will be set to the path of the picked primitive. If no primitive was picked, the pick status will be set to **PEXNoPick**.

The pick path can be used for echoing when the pick measure is created. However, it is not used as a start path from which to start picking.

The input record is a pointer to data used to update the pick measure and depends on the pick device type specified when the pick measure was created. If the pick device type is **PEXPickDeviceDCHitBox**, the input record should point to a data structure of type **PEXPDDCHitBox**. If the pick device type is **PEXPickDeviceNPCHitVolume**, the input record should point to a data structure of type **PEXPDNPCHitVolume**.

Data Structures

```
typedef XID             PEXPickMeasure;

typedef union {
    PEXPDNPCHitVolume       volume;
    PEXPDDCHitBox           box;
    PEXPickDataRecord       data;
} PEXPickRecord;

typedef PEXNPCSubVolume PEXPDNPCHitVolume;

typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXCoord;
```

```
typedef struct {
    PEXDeviceCoord2D    position;
    float               distance;
} PEXPDDCHitBox;

typedef struct {
    short               x;
    short               y;
} PEXDeviceCoord2D;

typedef struct {
    unsigned short      length;                 /* number of bytes in record */
    char                *record;
} PEXPickDataRecord;
```

Errors

BadPEXPath The specified path is invalid.

BadPEXPickMeasure
The specified pick measure resource identifier is invalid.

See Also

PEXFreePickMeasure

Chapter 12

PEX Fonts

The PEX font manipulation mechanisms are the same mechanism as those for X11 fonts. The PEX font resource is very similar to the font resource created and managed by X11. PEX fonts that are used with PEX stroke precision text contain more functionality than is currently found in X11 fonts. Specifically, stroke precision text requires scalable and rotatable text fonts. Because of the added capabilities of PEX fonts, PEX has defined its own PEX fonts open, close and query requests. However, the PEXlib functions correlate to the Xlib font functions.

PEX uses the X11 font path in order to locate PEX font files.

The first, last and default glyph fields are indices to the first, last and default glyphs of the font. The default glyph specifies the glyph that will be displayed when an undefined or non-existent glyph is used. If the default glyph specifies a non-existent glyph, nothing will be rendered for undefined or non-existent glyphs. If all glyphs exist is **True**, then all glyphs within the range of the first and last glyph have non-zero extents. The stroke font flag indicates whether the font is a PEX font and is provided so that the application can build font groups that all have the same text precision.

A font is not guaranteed to have any properties defined. Whether a property value is a signed or unsigned, and in what units it is specified must be derived from prior knowledge of the property. Many fonts will have at least a CHARSET_REGISTRY (e.g. "ISO8859") and a CHARSET_ENCODING (e.g. "1") property.

See also information provided in the X Logical Font Description (XLFD) Conventions defined by the X Consortium.

PEXUnloadFont - Unload PEX Font

Synopsis

void PEXUnloadFont(Display **display*, PEXFont *font*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
font	The resource identifier of the PEX font.

Returns

None

Description

This function deletes the association between the resource identifier and the PEX font. The PEX font itself will be freed when no other resource references it.

Data Structures

```
typedef XID              PEXFont;
```

Errors

BadPEXFont	The specified font resource identifier is invalid.

See Also

PEXLoadFont

PEXFreeFontInfo - Free Font Info Returned by PEXListFontsWithInfo and PEXQueryFont

Synopsis

void PEXFreeFontInfo(unsigned long *count*, PEXFontInfo **font_info*)

Arguments

count	The number of font info structures.
font_info	An array of font info structures.

Returns

None

Description

This function deallocates memory returned by **PEXListFontsWithInfo** and **PEXQueryFont**.

Errors

None

See Also

PEXListFontsWithInfo, **PEXQueryFont**

PEXFreeFontNames - Free Font Names Returned by PEXListFonts, PEXListFontsWithInfo

Synopsis

void PEXFreeFontNames(unsigned long *count*, char ***font_names*)

Arguments

count	The number of font names.
font_names	An array of font names (null-terminated strings).

Returns

None

Description

This function deallocates memory returned by **PEXListFonts** and **PEXListFontsWithInfo**.

Errors

None

See Also

PEXListFonts, **PEXListFontsWithInfo**

PEXListFonts - List PEX Fonts

Synopsis

char **PEXListFonts(Display **display*, char **pattern*, unsigned int *max_names*, unsigned long **count_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
pattern	A null-terminated string pattern.
max_names	The maximum number of names to be returned.
count_return	Returns the actual number of font names.

Returns

An array of font names (null-terminated strings); a null pointer if unsuccessful or if no PEX fonts supported.

Description

This function is like X11 **XListFonts** except only information about fonts that can support the full range of PEX text attributes is returned (i.e. only those fonts that are "PEX usable"). This list may or may not contain some of the same fonts returned by the X11 **XListFonts** function. This function returns a list of entries, each of which contains the name of a font matching the pattern. The pattern is a string that uses the ISO Latin-1 encoding and case is not significant in matching names. The '?' character (octal value 077) matches any single character, and the character '*' (octal value 052) matches any number of characters. The returned names are null-terminated, in lower-case and are also ISO Latin-1 encoded strings.

PEXlib allocates memory for the returned list of font names. **PEXFreeFontNames** should be called to deallocate the memory.

Errors

None

See Also

PEXListFontsWithInfo, **PEXFreeFontNames**

PEXListFontsWithInfo - List PEX Fonts With Information

Synopsis

char **PEXListFontsWithInfo(Display **display*, char **pattern*, unsigned int *max_names*, unsigned long **count_return*, PEXFontInfo ***info_return*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
pattern	A null-terminated string pattern.
max_names	The maximum number of names to be returned.
count_return	Returns the actual number of font names.
font_info	Returns an array of font info structures (one for each name).

Returns

An array of font names (null-terminated strings); a null pointer if unsuccessful or if no PEX fonts supported.

Description

This function is like X11 **XListFontsWithInfo** except that only information about fonts that can support the full range of PEX text attributes is returned (i.e. those fonts that are "PEX usable"). This list may or may not contain some of the same fonts returned the X11 **XListFontsWithInfo** function. This function returns a list of entries, each of which contains information about a font matching the pattern. The pattern is a string that uses the ISO Latin-1 encoding and case is not significant in matching names. The '?' character (octal value 077) matches any single character, and the character '*' (octal value 052) matches any number of characters. The returned names are null-terminated, in lower case and are also ISO Latin-1 encoded strings.

The information returned for each font is identical to what **PEXQueryFont** would return.

PEXlib allocates memory for the returned list of font names and font info. **PEXFreeFontNames** should be called to deallocate the memory for the font names. **PEXFreeFontInfo** should be called to deallocate the memory for the font information.

Data Structures

```
typedef struct {
    unsigned long      first_glyph;
    unsigned long      last_glyph;
    unsigned long      default_glyph;
    Bool               all_exist;
    Bool               stroke;
    unsigned short     count;                  /* number of properties */
    PEXFontProp        *props;
} PEXFontInfo;

typedef struct {
    Atom               name;
    unsigned long      value;
} PEXFontProp;
```

Errors

None

See Also

PEXListFonts, **PEXQueryFont**
PEXFreeFontNames, **PEXFreeFontInfo**

PEXLoadFont - Load PEX Font

Synopsis

PEXFont PEXLoadFont(Display **display*, char **font_name*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
font_name	The font name (null-terminated string).

Returns

The resource identifier of the loaded PEX font.

Description

This function loads the specified PEX font, if necessary. The font name should use the ISO Latin-1 encoding and case is not significant in matching names. PEX fonts are not associated with a particular screen, and can be used with any renderer or PHIGS workstation resource. An error will be generated if the specified font is not "PEX usable", that is, it is not capable of supporting the full range of PEX text attributes.

Data Structures

```
typedef XID              PEXFont;
```

Errors

BadAlloc	The server failed to allocate the resource.
BadPEXFont	The specified font name does not name a usable PEX font.

See Also

PEXUnloadFont

PEXQueryEncodedTextExtents - Query Encoded Text Extents

Synopsis

PEXTextExtent *PEXQueryEncodedTextExtents(Display **display*, XID *resource_id*, unsigned int *font_table_index*, int *path*, double *expansion*, double *spacing*, double *height*, int *halign*, int *valign*, unsigned long *count*, PEXListOfEncodedText **encoded_text*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id	The resource identifier of either a renderer, a workstation or a text font table.
font_table_index	The text font table index.
path	The text path (**PEXPathRight**, **PEXPathLeft**, **PEXPathUp**, **PEXPathDown**).
expansion	The text character expansion factor.
spacing	The text character spacing factor.
height	The text character height.
halign	The text horizontal text alignment (**PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignCenter**, **PEXHAlignRight**).
valign	The text vertical text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase**, **PEXVAlignBottom**).
count	The number of encoded text strings.
encoded_text	An array of encoded text strings.

Returns

An array of text extents; a null pointer if unsuccessful.

Description

This function is like **PEXQueryTextExtents**, except that multiple encoded text strings are specified. Each text string in the encoded text list has a character set, a character set width, an encoding state, and a list of characters. (See **PEXEncodedTextData**.)

The character set is an index into the current font group. Font groups have individual fonts which are numbered starting at one; a value of three would select the third font in the font group currently being used. If a character set is not available in the current font group then the default font group is used. If a character set value is not available in the default font group then the result is implementation-dependent. The character set width indicates the width or size of characters in the strings. Valid values for character set width are **PEXCSByte**, **PEXCSShort** and **PEXCSLong**. The encoding state is provided for use by the application and is not interpreted by the PEX server.

All other aspects of this function are the same as **PEXQueryTextExtents**.

Data Structures

```
typedef struct {
    unsigned short      count;                  /* number of encodings */
    PEXEncodedTextData  *encoded_text;
} PEXListOfEncodedText;

typedef struct {
    unsigned short      character_set;
    unsigned char       character_set_width;
    unsigned char       encoding_state;
    unsigned short      reserved;
```

```
    unsigned short      length;
    char                *ch;
} PEXEncodedTextData;

typedef struct {
    PEXCoord2D          lower_left;
    PEXCoord2D          upper_right;
    PEXCoord2D          concat_point;
} PEXTextExtent;

typedef struct {
    float               x;
    float               y;
} PEXCoord2D;
```

Errors

BadMatch — The specified resource identifier identifies a table of type other than a text font table.

BadValue — A specified value for one or text attributes is invalid, or the specified resource identifier does not identify a valid renderer, workstation or lookup table resource.

See Also

PEXQueryTextExtents

PEXQueryFont - Query PEX Font Information

Synopsis

PEXFontInfo *PEXQueryFont(Display **display*, PEXFont *font*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
font	The resource identifier of the font.

Returns

A pointer to the font info structure; a null pointer if unsuccessful.

Description

This function returns information about the specified PEX font. The information returned is identical to what **PEXListFontsWithInfo** would return.

PEXlib allocates memory for the returned list of font information. **PEXFreeFontInfo** should be called to deallocate the memory.

Data Structures

```
typedef XID             PEXFont;

typedef struct {
    unsigned long       first_glyph;
    unsigned long       last_glyph;
    unsigned long       default_glyph;
    Bool                all_exist;
    Bool                stroke;
    unsigned short      count;                  /* number of properties */
    PEXFontProp         *props;
} PEXFontInfo;

typedef struct {
    Atom                name;
    unsigned long       value;
} PEXFontProp;
```

Errors

BadPEXFont	The specified font resource identifier is invalid.

See Also

PEXQueryTextExtents - Query Text Extents

Synopsis

PEXTextExtent *PEXQueryTextExtents(Display **display*, XID *resource_id*, unsigned int *font_table_index*, int *path*, double *expansion*, double *spacing*, double *height*, int *halign*, int *valign*, unsigned long *count*, PEXStringData **text*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
resource_id	The resource identifier of either a renderer, a workstation or a text font table.
font_table_index	The text font table index.
path	The text path (**PEXPathRight**, **PEXPathLeft**, **PEXPathUp**, **PEXPathDown**).
expansion	The text character expansion factor.
spacing	The text character spacing factor.
height	The text character height.
halign	The text horizontal text alignment (**PEXHAlignNormal**, **PEXHAlignLeft**, **PEXHAlignCenter**, **PEXHAlignRight**).
valign	The text vertical text alignment (**PEXVAlignNormal**, **PEXVAlignTop**, **PEXVAlignCap**, **PEXVAlignHalf**, **PEXVAlignBase**, **PEXVAlignBottom**).
count	The number of text strings.
text	An array of text strings.

Returns

An array of text extents; a null pointer if unsuccessful.

Description

This function returns the text extents, in the local 2D text coordinate system, of each specified text string. If the resource identifier is a renderer or PHIGS workstation, then the text font table associated with the renderer or workstation is used. If the resource identifier is a text font lookup table, it is used directly. The specified font group provides the index of the entry that is to be used to obtain the font group. The first character set in the font will be used.

Stroke precision is assumed. The text position is (0,0) in the local 2D text coordinate system. The extent of each string is computed independent of the other strings.

If a specified font has no defined default glyph, then undefined glyphs are taken to have all zero metrics.

The extent data is returned in memory allocated by PEXlib. **XFree** should be called to deallocate the memory.

Data Structures

```
typedef struct {
    unsigned short      length;
    char                *ch;
} PEXStringData;

typedef struct {
    PEXCoord2D          lower_left;
    PEXCoord2D          upper_right;
    PEXCoord2D          concat_point;
```

```
} PEXTextExtent;

typedef struct {
    float               x;
    float               y;
} PEXCoord2D;
```

Errors

BadMatch The specified resource identifier identifies a table of type other than a text font table.

BadValue A specified value for one or text attributes is invalid, or the specified resource identifier does not identify a valid renderer, workstation or lookup table resource.

See Also

PEXQueryEncodedTextExtents

Chapter 13

PEX Escapes

PEXEscape and **PEXEscapeWithReply** are functions which provide a standard interface to implementation-dependent or vendor specific requests. **PEXSetEchoColor** is a convenience function for easy access to the one standard PEX escape.

PEXEscape - PEX Escape

Synopsis

void PEXEscape(Display **display*, unsigned long *escape_id*, int *length*, char **escape_data*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
escape_id	The escape identifier.
length	The length, in bytes, of data for the escape request.
escape_data	A pointer to data for the escape request.

Returns

None

Description

PEXEscape has an implementation-dependent effect. It is provided as a way for implementation-specific functionality to be accessed. The complete interface and behavior for each specified escape identifier should be specified by the individual PEX server implementation.

There is one standard PEX escape, see below.

If the specified escape identifier is not supported, a value error is generated.

Standard PEX Escape:

> **PEXEscapeSetEchoColor** is an escape to set the echo color for use by a renderer when echoing primitives. **PEXEscapeSetEchoColor** is the name of the escape identifier and is passed as the escape identifier to **PEXEscape.** The escape data for **PEXEscapeSetEchoColor** consists of a renderer resource identifier (XID) followed by a color specifier (**PEXColorSpecifier**). See the **PEXSetEchoColor** function for further description of this standard escape.

Errors

Escape-dependent	See documentation provided with the individual PEX server implementation.
BadValue	The specified escape identifier is unsupported.

See Also

PEXEscapeWithReply, **PEXSetEchoColor**

PEXEscapeWithReply - PEX Escape With Reply

Synopsis

char *PEXEscapeWithReply(Display **display*, unsigned long *escape_id*, int *length*, char **escape_data*, unsigned long **reply_length_return*)

Arguments

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

escape_id The escape identifier.

length The length, in bytes, of data for the escape request.

escape_data A pointer to data for the escape request.

reply_length_return
Returns the length, in bytes, of the reply data.

Returns

A pointer to the escape reply data; a null pointer if unsuccessful.

Description

PEXEscapeWithReply has an implementation-dependent effect. It is similar to **PEXEscape** except it has return data. It is provided as a way for implementation-specific functionality to be accessed. The complete interface and behavior for each specified escape identifier should be specified by the individual PEX server implementation.

There are no standard PEX escapes with replies.

If the specified escape identifier is not supported, a value error is generated.

The reply data is returned in memory allocated by PEXlib. **XFree** should be called to deallocate the memory.

Errors

Escape-dependentSee documentation provided with the individual PEX server implementation.

BadValue The specified escape identifier is unsupported.

See Also

PEXEscape

PEXSetEchoColor - PEX Escape to set Echo Color for Specified Renderer

Synopsis

void PEXSetEchoColor(Display **display*, PEXRenderer *renderer*, int *color_type*, PEXColor **color*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
renderer	The resource identifier of a renderer.
color_type	The type of color (**PEXColorTypeIndexed**, **PEXColorTypeRGB**, **PEXColorTypeCIE**, **PEXColorTypeHSV**, **PEXColorTypeHLS**, **PEXColorTypeRGB8**, **PEXColorTypeRGB16**).
color	The echo color.

Returns

None

Description

This function is a convenient way to access the **PEXEscapeSetEchoColor** escape to set the echo color for use by a renderer when echoing primitives.

A renderer's echo color can be changed at any time during rendering and is not inquirable through the renderer's modification dynamics (see **PEXGetRendererDynamics**). Lighting and shading for echoed primitives is implementation-dependent. A renderer's echo color can not be inquired and the default value is implementation-dependent. Support of this escape is inquirable via **PEXGetEnumTypeInfo**.

Data Structures

```
typedef XID             PEXRenderer;

typedef union {
    PEXColorIndexed     indexed;
    PEXColorRGB         rgb;
    PEXColorHSV         hsv;
    PEXColorHLS         hls;
    PEXColorCIE         cie;
    PEXColorRGB8        rgb8;
    PEXColorRGB16       rgb16;
} PEXColor;

typedef struct {
    PEXTableIndex       index;
    unsigned short      reserved;
} PEXColorIndexed;

typedef struct {
    float               red;
    float               green;
    float               blue;
} PEXColorRGB;

typedef struct {
```

```
        float               hue;
        float               saturation;
        float               value;
    } PEXColorHSV;

    typedef struct {
        float               hue;
        float               lightness;
        float               saturation;
    } PEXColorHLS;

    typedef struct {
        float               x;
        float               y;
        float               z;
    } PEXColorCIE;

    typedef struct {
        unsigned char       red;
        unsigned char       green;
        unsigned char       blue;
        unsigned char       reserved;
    } PEXColorRGB8;

    typedef struct {
        unsigned short      red;
        unsigned short      green;
        unsigned short      blue;
        unsigned short      reserved;
    } PEXColorRGB16;
```

Errors

BadPEXColorType
The specified color type is invalid or unsupported.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadValue The **PEXEscapeSetEchoColor** escape is unsupported.

See Also

PEXEscape

Chapter 14

Output Command Encode and Decode

Some applications need to generate protocol-formatted output commands for use in archives or client-side data storage. Likewise, the application may need to decode the protocol-formatted output commands returned from functions like **PEXFetchElements**. The functions described in this chapter provide for these operations.

For complete information on protocol format, see the *PEX Protocol Specification* and the *PEX Protocol Encoding*.

For the definition of the **PEXOCData** data structure, see the PEXlib.h header file.

PEXCountOCs - Count Output Commands in an Encoded List

Synopsis

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

Arguments

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.

Errors

None

See Also

PEXDecodeOCs

PEXDecodeOCs - Decode Output Commands

Synopsis

PEXOCData *PEXDecodeOCs(int *float_format*, unsigned long *oc_count*, unsigned long *length*, char **encoded_ocs*)

Arguments

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 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. **PEXFreeOCData** 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.

PEXCountOCs can be used to determine the number of output commands in the encoded list if that information is not already available.

Errors

None

See Also

PEXEncodeOCs, **PEXCountOCs**

PEXEncodeOCs - Encode Output Commands

Synopsis

char *PEXEncodeOCs(int *float_format*, unsigned long *oc_count*, PEXOCData **oc_data*, unsigned long **length_return*)

Arguments

float_format	The floating point format for 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 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 deallocate 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.

Errors

None

See Also

PEXDecodeOCs

PEXFreeOCData - Deallocate OC Data

Synopsis

void PEXFreeOCData(unsigned long *count*, PEXOCData **oc_data*)

Arguments

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

Returns

None

Description

This function deallocates memory allocated by PEXlib to hold decoded output command data.

Errors

None

See Also

PEXDecodeOCs

Chapter 15

Sending Encoded Output Commands

Some applications have access to encoded output commands, such as for archives or client-side data storage obtained via **PEXEncodeOCs** or **PEXFetchElements**. The functions described in this chapter provide a way for the application to send this data directly instead of decoding the data and calling the appropriate output primitive or attribute functions.

There are two ways provided for sending encoded output commands. The first is **PEXSendOCs** which is useful if the application has the complete list of encoded output commands available in contiguous memory. The second is a set of functions: **PEXStartOCs**, **PEXCopyBytesToOC**, **PEXGetOCAddr**, **PEXFinishOCs**. These are useful if the application has data which is already encoded or which the application wishes to encode "on-the-fly", but which is not in contiguous memory. **PEXCopyBytesToOC** can be used to have PEXlib copy individual pieces which are in contiguous memory. **PEXGetOCAddr** can be used to get an address where the application can copy data on its own. Both of these functions copy data directly into the transport buffer, possibly avoiding one data copy from the application data into a format necessary to directly call the PEXlib output command functions.

None of these functions will convert floating point formats. It is up to the application to determine which formats are supported by the PEX server and do any necessary conversion on the data. For example, if the PEX server does not support the floating point format native to the application's machine, the application will have to convert any native float values into a format supported by the PEX server.

For complete information on protocol format, see the *PEX Protocol Specification* and the *PEX Protocol Encoding*.

PEXCopyBytesToOC - Copy Encoded Output Commands

Synopsis

void PEXCopyBytesToOC(Display **display*, int *length*, char **data*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
length	The number of bytes to copy.
data	A pointer to the output command data.

Returns

None

Description

This function copies the specified number of bytes of data to the transport buffer.

It is recommended that the number of bytes be a multiple of four as the protocol format requires output commands to be aligned on four-byte boundaries. It is the application's responsibility to ensure that alignment restrictions are met.

PEXStartOCs must be called prior to this.

Errors

None

See Also

PEXStartOCs, **PEXFinishOCs**, **PEXGetOCAddr**

PEXGetOCAddr - Get Address For Encoded Output Commands

Synopsis

char *PEXGetOCAddr(Display **display*, int *length*)

Arguments

display	A pointer to a display structure returned by a successful **XOpenDisplay** call.
length	The number of bytes of data to be written by the application.

Returns

A pointer to memory where the application can write output command data; a null pointer if unsuccessful.

Description

This function returns a memory address to the specified number of bytes in the transport buffer where the application can write data.

The pointer returned is valid only until the next **PEXGetOCAddr** or **PEXCopyBytesToOC** is called.

An attempt to request more bytes than remaining in the transport buffer, or more bytes than returned by **PEXGetOCAddrMaxSize**, will result in an unsuccessful return value.

PEXStartOCs must be called prior to this.

DO NOT attempt to deallocate or free memory at the address returned by this function.

Errors

None

See Also

PEXStartOCs, **PEXFinishOCs**, **PEXCopyBytesToOC**, **PEXGetOCAddrMaxSize**

PEXFinishOCs - Finish Encoded Output Commands

Synopsis

void PEXFinishOCs(Display **display*)

Arguments

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

Returns

None

Description

This function should be called to complete the sending of encoded output commands. The display connection is unlocked.

Errors

None

See Also

PEXStartOCs, **PEXCopyBytesToOC**, **PEXGetOCAddr**

PEXSendOCs - Send Encoded Output Commands

Synopsis

void PEXSendOCs(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *float_format*, unsigned long *oc_count*, unsigned int *length*, char **encoded_ocs*)

Arguments

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**).
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 encoded output commands.
length	The length, in bytes, of the encoded output commands.
encoded_ocs	A pointer to the encoded output commands.

Returns

None

Description

This function sends encoded output commands to the specified PEX server display.

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 type 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.

Errors

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXDecodeOCs, **PEXEncodeOCs**

PEXStartOCs - Start Encoded Output Commands

Synopsis

Status PEXStartOCs(Display **display*, XID *resource_id*, PEXOCRequestType *req_type*, int *float_format*, int *oc_count*, int *word_count*)

Arguments

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**).
float_format	The floating point format of the output command data (**PEXIEEE_754_32**, **PEXDEC_F_Floating**, **PEXIEEE_754_64**, **PEXDEC_D_Floating**).
oc_count	The number of output commands to be sent.
word_count	The number of four-byte words of data for the total size of the output commands.

Returns

Zero is unsuccessful, non-zero otherwise.

Description

This function locks the display. Only **PEXCopyBytesToOC** or **PEXGetOCAddr** may be called between pairs of **PEXStartOCs** and **PEXFinishOCs**. Do not call anything else that may lock the display as this will result in deadlock.

The first output command is guaranteed to start on a four-byte boundary. Output command data may be copied into the transport buffer by calling **PEXCopyBytesToOC**. Output command data may be written directly by the application by calling **PEXGetOCAddr** to get a pointer to memory in the transport buffer.

PEXFinishOCs must be called after all the data has been specified.

The application is responsible for writing valid protocol and the correct number of words requested.

If the requested number of words is too large for the display connection (each server has a maximum request size), the function will return unsuccessfully. If this occurs, and the number of output commands was greater than one, the application should try specifying the data for a single output command at a time. If the size of a single output command is too large for the display connection, the function will return unsuccessfully.

Errors

BadPEXFloatingPointFormat
The specified floating point format is invalid or unsupported.

BadPEXOutputCommand
The output command contains an invalid value.

BadPEXRenderer The specified renderer resource identifier is invalid.

BadPEXStructure The specified structure resource identifier is invalid.

See Also

PEXFinishOCs, **PEXCopyBytesToOC**, **PEXGetOCAddr**

PEXGetOCAddrMaxSize - Macro to Obtain the Maximum Size for PEXGetOCAddr

Synopsis

PEXGetOCAddrMaxSize(*display*)

Arguments

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

Description

This macro evaluates to the maximum size for the length parameter of **PEXGetOCAddr**.

Errors

None

See Also

PEXGetOCAddr

Chapter 16

PEXlib Utilities

This chapter gives the PEXlib utility functions. These functions can be used to create transformation matrices for modeling and viewing, and to apply those matrices to points and vectors. Utilities are also provided to compute the geometric normals of multi-faceted output primitives.

There are four groups of functions:

- Modeling Transformation Utilities
- Viewing Transformation Utilities
- Miscellaneous Transformation Utilities
- Utilities for Computing Geometric Normals

16.1. Modeling Transformation Utilities

Each function in this group computes a transformation matrix that performs a common modeling operation, such as a rotation or translation, or composes two transforms. Included in this set are the following functions:

PEXRotate
PEXRotate2D
PEXRotateGeneral
PEXScale
PEXScale2D
PEXTranslate
PEXTranslate2D
PEXMatrixMult
PEXMatrixMult2D
PEXBuildTransform
PEXBuildTransform2D

16.1.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* coordinates */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXCoord;

      typedef struct {
          float               x;
          float               y;
      } PEXCoord2D;

/* vectors */

      typedef struct {
          float               x;
```

```
        float               y;
        float               z;
    } PEXVector;

    typedef struct {
        float               x;
        float               y;
    } PEXVector2D;

/* matrices */

    typedef float           PEXMatrix[4][4];
    typedef float           PEXMatrix3x3[3][3];
```

PEXRotate

Synopsis

int PEXRotate(int *axis*, double *angle*, PEXMatrix *matrix_return*)

Arguments

axis	one of **PEXXAxis**, **PEXYAxis**, **PEXZAxis**
angle	angle of the rotation in radians
matrix_return	matrix into which rotation matrix is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadAxis - invalid axis value specified

Description

This function creates a rotation matrix about the specified axis.

The resulting matrix rotates by the angle specified in radians about the origin.

The function returns unsuccessfully if axis is not one defined.

Errors

None

See Also

PEXRotate2D, **PEXRotateGeneral**

PEXRotate2D

Synopsis

void PEXRotate2D(double *angle*, PEXMatrix3x3 *matrix_return*)

Arguments

angle	angle of the rotation in radians
matrix_return	matrix into which rotation matrix is stored

Returns

None.

Description

The resulting matrix rotates by the angle specified in radians about the origin.

Errors

None

See Also

PEXRotate, **PEXRotateGeneral**

PEXRotateGeneral

Synopsis

int PEXRotateGeneral(PEXCoord **point1*, PEXCoord **point2*, double *angle*, PEXMatrix *matrix_return*)

Arguments

point1	endpoint of line segment defining rotation axis
point2	endpoint of line segment defining rotation axis
angle	angle of the rotation in radians
matrix_return	matrix into which rotation matrix is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadAxis - the endpoints are coincident

Description

This routine formats a matrix for a right-handed rotation about an arbitrary axis. The rotation axis is defined by the line segment joining the two points. The angle is in radians and is measured counter-clockwise when looking from the first point to second point along the specified axis.

Returns unsuccessfully if the points are coincident.

Errors

None

See Also

PEXScale

Synopsis

void PEXScale(PEXVector **scale_vector*, PEXMatrix *matrix_return*)

Arguments

scale_vector	vector containing the X, Y and Z scale factors
matrix_return	matrix into which rotation matrix is stored

Returns

None

Description

Creates a scale matrix given the scale vector.

Errors

None

See Also

PEXScale2D

PEXScale2D

Synopsis

void PEXScale2D(PEXVector2D **scale_vector*, PEXMatrix3x3 *matrix_return*)

Arguments

scale_vector	vector containing the X and Y scale factors
matrix_return	matrix into which rotation matrix is stored

Returns

None

Description

Creates a 3X3 scale matrix given the scale vector.

Errors

None

See Also

PEXScale

PEXTranslate

Synopsis

void PEXTranslate(PEXVector **trans_vector*, PEXMatrix *matrix_return*)

Arguments

trans_vector	vector containing the X, Y and Z translation factors
matrix_return	matrix into which rotation matrix is stored

Returns

None

Description

Creates a translation matrix that translates objects by the given translation vector.

Errors

None

See Also

PEXTranslate2D

PEXTranslate2D

Synopsis

void PEXTranslate2D(PEXVector2D **trans_vector*, PEXMatrix3x3 *matrix_return*)

Arguments

trans_vector	vector containing the X and Y translation factors
matrix_return	matrix into which rotation matrix is stored

Returns

None

Description

Creates a 3X3 translation matrix that translates objects by the given translation vector.

Errors

None

See Also

PEXTranslate

PEXMatrixMult

Synopsis

void PEXMatrixMult(PEXMatrix *matrix1*, PEXMatrix *matrix2*, PEXMatrix *matrix_return*)

Arguments

matrix1	first matrix to be multiplied
matrix2	second matrix to be multiplied
matrix_return	matrix into which result is stored

Returns

None

Description

Performs a matrix multiplication: matrix = matrix1 X matrix2

If the return matrix is the same as one of the input matrices, the function will overwrite the input values with the multiplied values.

Errors

None

See Also

PEXMatrixMult2D

PEXMatrixMult2D

Synopsis

void PEXMatrixMult2D(PEXMatrix3x3 *matrix1*, PEXMatrix3x3 *matrix2*, PEXMatrix3x3 *matrix_return*)

Arguments

matrix1	first matrix to be multiplied
matrix2	second matrix to be multiplied
matrix_return	matrix into which result is stored

Returns

None

Description

Performs a 3X3 matrix multiplication: matrix = matrix1 X matrix2.

If the return matrix is the same as one of the input matrices, the function will overwrite the input values with the multiplied values.

Errors

None

See Also

PEXMatrixMult

PEXBuildTransform

Synopsis

void PEXBuildTransform(PEXCoord **fixed_point*, PEXVector **trans_vector*, double *angle_x*, double *angle_y*, double *angle_z*, PEXVector **scale_vector*, PEXMatrix *matrix_return*)

Arguments

fixed_point	origin for scaling and rotation
trans_vector	translation vector
angle_x	angle of rotation about X axis, in radians
angle_y	angle of rotation about Y axis, in radians
angle_z	angle of rotation about Z axis, in radians
scale_vector	vector of scale factors for X, Y and Z
matrix_return	matrix into which result is stored

Returns

None

Description

This function builds a transformation matrix that scales by the values in the scale vector about the fixed point, rotates about the X, Y and Z axes using the fixed point as the center of rotation and then translates according to translation vector, in that order.

Errors

None

See Also

PEXBuildTransform2D

PEXBuildTransform2D

Synopsis

void PEXBuildTransform2D(PEXCoord2D **fixed_point*, PEXVector2D **trans_vector*, double *angle_z*, PEXVector2D **scale_vector*, PEXMatrix3x3 *matrix_return*)

Arguments

fixed_point	origin for scaling and rotation
trans_vector	translation vector
angle_z	angle of rotation about Z axis, in radians
scale_vector	vector of scale factors for X and Y axes
matrix_return	matrix in which result is stored

Returns

None

Description

This function builds a 3X3 transformation matrix that scales by the values in the scale vector about the fixed point, rotates about Z axis using the fixed point as the center of rotation and then translates according to translation vector, in that order.

Errors

None

See Also

PEXBuildTransform

16.2. Viewing Transformation Utilities

These functions compute viewing transforms for the PHIGS viewing model and a more simple viewing model. The functions using the PHIGS viewing model are:

PEXViewOrientationMatrix
PEXViewOrientationMatrix2D
PEXViewMappingMatrix
PEXViewMappingMatrix2D

Those for the simple viewing model are:

PEXLookAtViewMatrix
PEXPolarViewMatrix
PEXOrthoProjMatrix
PEXPerspProjMatrix

The matrices returned by these functions can be passed to PEXlib as viewing transforms.

16.2.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* coordinates */

      typedef struct {
          float              x;
          float              y;
          float              z;
      } PEXCoord;

      typedef struct {
          float              x;
          float              y;
      } PEXCoord2D;

/* vectors */

      typedef struct {
          float              x;
          float              y;
          float              z;
      } PEXVector;

      typedef struct {
          float              x;
          float              y;
      } PEXVector2D;

/* matrices */

      typedef float          PEXMatrix[4][4];
      typedef float          PEXMatrix3x3[3][3];
```

PEXViewOrientationMatrix

Synopsis

int PEXViewOrientationMatrix(PEXCoord **vrp*, PEXVector **vpn*, PEXVector **vup*, PEXMatrix *matrix_return*)

Arguments

vrp	view reference point
vpn	view plane normal
vup	view up vector
matrix_return	matrix into which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector - either vpn or vup is zero length
PEXBadVectors - vup is parallel to vpn

Description

This function creates a view orientation matrix that transforms world coordinates (WC) to view reference coordinates (VRC). This matrix is used in conjunction with a view mapping matrix as the viewing matrices for a designated view.

The view reference point (VRP) defines the point in world coordinate space that is to be used as the origin of the view reference coordinate system.

The view plane normal (VPN) is a 3D vector defined in world coordinates relative to the view reference point. This gives the direction of the positive Z axis of VRC.

The view up vector (VUP) is a 3D vector defined in world coordinates relative to the view reference point. The projection of this vector onto the plane through the view reference point and perpendicular to the view plane normal defines the Y axis of VRC.

The X axis of VRC is defined such that the VRC system forms a right-handed coordinate system.

Errors

None

See Also

PEXViewOrientationMatrix2D, **PEXViewMappingMatrix**, **PEXViewMappingMatrix2D**

PEXViewOrientationMatrix2D

Synopsis

int PEXViewOrientationMatrix2D(PEXCoord2D **vrp*, PEXVector2D **vup*, PEXMatrix3x3 *matrix_return*)

Arguments

vrp	view reference point
vup	view up vector
matrix_return	matrix in which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector - vup is zero length.

Description

This function creates a view orientation matrix that transforms world coordinates (WC) to view reference coordinates (VRC). This matrix is used in conjunction with a view mapping matrix as the viewing matrices for a designated view.

The view reference point (VRP) defines the point in world coordinate space that is to be used as the origin of the view reference coordinate system. The point is in the WC z=0 plane.

The view plane normal (VPN) is in the direction of the positive Z axis of WC.

The view up vector (VUP) is a 2D vector in the WC z=0 plane. It determines the positive Y axis of VRC. It's defined in world coordinates relative to the view reference point.

The X axis of VRC is defined such that the VRC system forms a right-handed coordinate system.

Errors

None

See Also

PEXViewOrientationMatrix, **PEXViewMappingMatrix**, **PEXViewMappingMatrix2D**

PEXViewMappingMatrix

Synopsis

int PEXViewMappingMatrix(PEXCoord2D **frame*, PEXNPCSubVolume **viewport*, int *perspective*, PEXCoord **prp*, double *view_plane*, double *back_plane*, double *front_plane*, PEXMatrix *matrix_return*)

Arguments

frame	array of 2 2D VRC locations which mark a rectangle in the view plane
viewport	NPC viewport into which the frame gets mapped
perspective	flag to indicate whether a perspective view is desired, a value of **True** requests that perspective be applied
prp	projection reference point
view_plane	VRC position of view plane w.r.t. the VRP
back_plane	VRC position of the back plane w.r.t. the VRP
front_plane	VRC position of the front plane w.r.t. the VRP
matrix_return	matrix into which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits
PEXBadViewport
PEXBadPlanes
PEXBadPRP

Description

This function creates a view mapping matrix that transforms a volume specified in view reference coordinates (VRC) to a volume in normalized projection coordinates (NPC). This matrix is used in conjunction with a view orientation matrix as the viewing matrices for a designated view.

The axes of VRC form a right-handed coordinate system. The z axis is along the VPN (view plane normal, see **PEXViewOrientation**), The Y axis is fixed by VUP, and the X axis is determined so that the three axes form a right- handed coordinate system.

The front plane, back plane, and view plane all define planes in VRC parallel to the VRC x-y plane. The location of front_plane and back_plane along the z axis of VRC defines the front and back of the volume of VRC that will be mapped to the specified NPC viewport. The view plane locates the view frame or "window" on the VRC z axis. The two points in frame determine the size of the view window by specifying lower left (frame[0]) and upper right (frame[1]) x and y VRC points of the window on the view plane. These values taken together establish the volume of VRC space that is mapped into the NPC viewport.

The type of projection may be parallel or perspective. The projection reference point (PRP) orients the projectors defining the surfaces of the view volume. If perspective indicator is **False**, then the projection type is parallel and the projectors are all parallel to the vector joining the projection reference point and the center of the view window (located on the view plane). If perspective is **True**, then the projectors all converge at the projection reference point. Thus, the view volume is a parallelpiped for parallel views, and a truncated pyramid for perspective views.

When specifying NPC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Data Structures

```
typedef struct {
    PEXCoord            min;
    PEXCoord            max;
} PEXNPCSubVolume;
```

See also the *Common Data Structures* section.

Errors

None

See Also

PEXViewMappingMatrix2D, **PEXViewOrientationMatrix**, **PEXViewOrientationMatrix2D**

PEXViewMappingMatrix2D

Synopsis

int PEXViewMappingMatrix2D(PEXCoord2D **frame*, PEXCoord2D **viewport*, PEXMatrix3x3 *matrix_return*)

Arguments

frame	array of 2 2D VRC locations which mark a rectangle to be mapped into the NPC subvolume.
viewport	array of 2 2D NPC coordinates with z = 0 specifying lower left and upper right of the "viewport".
matrix_return	matrix in which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits
PEXBadViewport

Description

This function is a 2D shorthand version of **PEXViewMappingMatrix** with the following parameters assumed:

front plane distance = 1
back plane distance = 0
view plane distance = 0
viewport z range = [0,1]
projection type = parallel
x-y coordinates of projection reference point = center of view window,
z coordinate = 1.0.

When specifying NPC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXViewMappingMatrix, **PEXViewOrientationMatrix**, **PEXViewOrientationMatrix2D**

PEXLookAtViewMatrix

Synopsis

int PEXLookAtViewMatrix(PEXCoord **from*, PEXCoord **to*, PEXVector **up*, PEXMatrix *matrix_return*)

Arguments

from	Viewing position, in world coordinates.
to	Look at position, in world coordinates.
up	Vector representing the "up" direction, in world coordinates.
matrix_return	matrix in which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadVectors - the from and to arguments are equal, or the line between them is parallel with the up vector
PEXBadVector - up is zero length

Description

This function creates a view orientation transform that defines the viewing direction and orientation. It is a slightly more intuitive interface to **PEXViewOrientationMatrix**.

The "from" position defines the viewpoint, and the "to" position specifies the point being viewed. These two parameters together define the view reference point (the VRC origin) and the view plane normal of **PEXViewOrientationMatrix**. The view reference point is the "to" point; the view plane normal is the vector from "to" to "from".

The view up vector is a 3D vector defined in world coordinates relative to the "to" point. The projection of this vector onto the plane through the "to" point and perpendicular to the view plane normal defines the Y axis of VRC.

Errors

None

See Also

PEXPolarViewMatrix, **PEXViewOrientationMatrix**, **PEXViewMappingMatrix**

PEXPolarViewMatrix

Synopsis

int PEXPolarViewMatrix(PEXCoord **from*, double *distance*, double *azimuth*, double *altitude*, double *twist*, PEXMatrix *matrix_return*)

Arguments

from	The viewing position.
distance	The distance between the ‘from’ position and the position being viewed.
azimuth	The angle in the x,z plane from the +z axis to the line of sight, in radians. Positive values are counter-clockwise when viewed from the positive ‘y’ axis.
altitude	The angular inclination of the line of sight from the ‘x’,‘z’ plane. The ‘altitude’ argument is the angle in radians. Positive values are towards the positive ‘y’ axis.
twist	The up direction of the view, given as an angle, in radians, about the line of sight. Positive values of twist are in the counter-clockwise direction.
matrix_return	matrix in which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadDistance

Description

This routine formats a polar view orientation matrix. This routine is similar to **PEXLookAtViewMatrix**, except that the viewing parameters are specified in spherical coordinates. The "from" position defines one end of the view plane normal; the position indicated by "distance", "azimuth", and "altitude" define the base of the view plane normal, and the origin of VRC.

The view is defined with respect to the ‘from’ position (the viewing position) and the distance between it and the position being viewed. The azimuth angle specifies the direction of the line of sight going toward the position being viewed. Positive values of azimuth are counter-clockwise when viewed from the positive y axis.

The azimuth and altitude angles apply to the coordinate system with ‘from’ at the origin and the line of sight emanating from it. The azimuth specifies the angle between the line of sight and the +z axis, and the altitude defines the angle between it and the x,z plane.

When applied, the transformation places the viewing position at the origin, aligns the viewpoint with the +z axis, applies any twist to the coordinates, and then places the viewed point at the origin.

Errors

None

See Also

PEXLookAtViewMatrix, **PEXViewOrientationMatrix**, **PEXViewMappingMatrix**

PEXOrthoProjMatrix

Synopsis

int PEXOrthoProjMatrix(double *height*, double *aspect*, double *near*, double *far*, PEXMatrix *matrix_return*)

Arguments

height	The height of the orthographic viewing box.
aspect	The aspect ratio (width/height) of the orthographic viewing box.
near	The distance, in view reference coordinates, from the VRC origin to the front clipping plane.
far	The distance, in view reference coordinates, from the VRC origin to the back clipping plane.
matrix_return	matrix in which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits - the viewing box depth, width, or height is zero.

Description

This routine formats a view mapping matrix.

A projection matrix defines a visible region of the coordinate space. An orthographic projection defines the visible region as a box specified by its height, width (the height multiplied by the aspect), and its near and far boundaries.

The reference point for the projection is the origin of VRC; the near and far clipping planes are defined with respect to it. The height is defined in view reference coordinates. Clipping at the planes is controlled by the clipping flags in the selected view table entry.

Errors

None

See Also

PEXLookAtViewMatrix, **PEXViewOrientationMatrix**, **PEXViewMappingMatrix**, **PEXPerspProjMatrix**

PEXPerspProjMatrix

Synopsis

int PEXPerspProjMatrix(double *fovy*, double *distance*, double *aspect*, double *near*, double *far*, PEXMatrix *matrix_return*)

Arguments

fovy	Field of view (in radians) in the horizontal direction.
distance	The distance to the eye-point.
aspect	The aspect ratio (width/height) of the perspective viewing frustum.
near	The distance to the near clipping plane.
far	The distance to the far clipping plane.
matrix_return	matrix in which result is stored

Returns

Zero if successful; otherwise, one of the following:

PEXBadLimits - near <= far, fovy = 0, aspect = 0, or distance <= near

Description

This routine formats a view mapping matrix.

A projection matrix defines the visible region of the coordinate space. A perspective projection defines the visible region as a truncated pyramid or frustum. The amount of perspective in the projection is specified by the field of view argument, ‘fovy’. The perspective increases as the angle increases to a value of pi radians.

The distance between the eye-point and the origin is specified by the distance. If the application program calls **PEXLookAtViewMatrix** to calculate the view orientation matrix, the distance is typically the distance between the ‘from’ and ‘to’ points specified to that routine. If the application program calls **PEXPolarViewMatrix** to calculate the view orientation matrix, the distance is typically the same distance specified to that routine.

The height of the frustum at the near clipping plane is determined by ‘fovy’ and the distance to the near plane. The width of the frustum is determined from the aspect ratio.

The reference point for the projection is the origin of VRC; the near and far clipping planes are defined with respect to it. Clipping at the planes is controlled by the clip flags in the selected view table entry.

It is useful to think of **PEXPerspProjMatrix** as defining a camera. The object being viewed is defined near the origin. The lens is defined by ‘fovy’; a larger value of ‘fovy’ defines a wide angle lens. For those who wish to keep the height at the near plane constant and automatically back up the camera to frame the subject, the relationship between the field of view, the eye distance, which is the distance between the eye-point and the near plane, and the height, which is the height at the near plane, is:

tan (fovy/2) = ((height/2) / eye_distance)

For example, if a unit cube is being viewed, a "look at" view with the ‘to’ point at the center of the cube or a "polar" view with the viewed point at the center of the cube, places the cube at the origin. A matrix created by **PEXPerspProjMatrix** with aspect = 1, near = 0.5, and far = -0.5 makes the entire cube visible, with the field of view and distance controlling the amount of perspective applied.

Errors

None

See Also

PEXLookAtViewMatrix, **PEXViewOrientationMatrix**, **PEXViewMappingMatrix**, **PEXOrthoProjMatrix**

16.3. Miscellaneous Transformation Utilities

This set of functions computes various useful transforms and applies transforms to points and vectors. Included in this set are:

PEXTransformPoints
PEXTransformPoints2D
PEXTransformPoints4D
PEXTransformPoints2DH
PEXTransformVectors
PEXTransformVectors2D
PEXNormalizeVectors
PEXNormalizeVectors2D
PEXInvertMatrix
PEXInvertMatrix2D
PEXIdentityMatrix
PEXIdentityMatrix2D
PEXNPCToXCTransform
PEXNPCToXCTransform2D
PEXXCToNPCTransform
PEXXCToNPCTransform2D
PEXMapXCToNPC
PEXMapXCToNPC2D

PEXMapXCToNPC or **PEXMapXCToNPC2D** can be used to find the PEX view containing a drawable coordinate (XC) point. **PEXNPCToXCTransform** and **PEXNPCToXCTransform2D** computes the matrix representing the transformation from the NPC subvolume through the DC viewport and on to drawable coordinates. **PEXXCToNPCTransform** and **PEXXCToNPCTransform2D** computes the inverse of this transformation. These functions can be used to transform points from NPC to XC and back, using the same transformation used by a PEX renderer or a PHIGS workstation resource.

As an example of using these functions, a list of modeling coordinate points could be transformed to drawable coordinates by the following sequence:

- Concatenate the local and global modeling transforms

  PEXMatrixMult(C = G x L)
- Concatenate the viewing transforms

  PEXMatrixMult(V = M x O)
- Concatenate the composite modeling and viewing transforms

  PEXMatrixMult(A = V x C)
- Compute the NPC to XC transform

  PEXNPCToXCTransform(= W)
- Concatenate that with the matrix from step c

  PEXMatrixMult(T = W x A)
- Apply the composite transform

  PEXTransformPoints(P’ = T x P)

By performing this operation in several steps, the application can precompute the matrices and apply them to several sets of points, at the same time skipping any steps that are unimportant.

16.3.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* coordinates */
```

```
    typedef struct {
        float               x;
        float               y;
        float               z;
    } PEXCoord;

    typedef struct {
        float               x;
        float               y;
    } PEXCoord2D;

    typedef struct {
        float               x;
        float               y;
        float               z;
        float               w;
    } PEXCoord4D;

/* vectors */

    typedef struct {
        float               x;
        float               y;
        float               z;
    } PEXVector;

    typedef struct {
        float               x;
        float               y;
    } PEXVector2D;

/* matrices */

    typedef float           PEXMatrix[4][4];
    typedef float           PEXMatrix3x3[3][3];

/* device coordinates, viewports and views */

    typedef struct {
        PEXCoord            min;
        PEXCoord            max;
    } PEXNPCSubVolume;

    typedef struct {
        short               x;
        short               y;
        float               z;
    } PEXDeviceCoord;

    typedef struct {
        short               x;
        short               y;
    } PEXDeviceCoord2D;

    typedef struct {
        unsigned short      clip_flags;
        unsigned short      reserved;
```

```
    PEXNPCSubVolume     clip_limits;
    PEXMatrix           orientation;
    PEXMatrix           mapping;
} PEXViewEntry;
```

PEXTransformPoints

Synopsis

int PEXTransformPoints(PEXMatrix *transform*, int *count*, PEXCoord **points*, PEXCoord **points_return*)

Arguments

transform	The transformation matrix to apply to the points
count	The number of points to transform
points	A pointer to an array of 3D points to transform
points_return	A pointer to an array in which to store the transformed points

Returns

Zero if successful; otherwise, one of the following:

PEXBadHomoCoord - one or more of the transformed points has a homogeneous coordinate of 0.

Description

This function applies the specified homogeneous transformation matrix to the list of points. In applying the transformation, the points are first converted to homogeneous points by assigning them a homogeneous coordinate of 1. The transformation is then applied:

P' = TxP

Where P is the point, treated as a column vector, and T is the transformation matrix. The points are then mapped to 3D by dividing their first three coordinates by the computed homogeneous coordinate.

If the function returns unsuccessfully, all points other than those with a homogeneous coordinate of 0 will be transformed and returned.

If the return array is the same as the input array, the function will overwrite the input values with the transformed values.

Errors

None

See Also

PEXTransformPoints2D, **PEXTransformPoints4D**, **PEXTransformPoints2DH**

PEXTransformPoints2D

Synopsis

int PEXTransformPoints2D(PEXMatrix3x3 *transform*, int *count*, PEXCoord2D **points*, PEXCoord2D **points_return*)

Arguments

transform	The transformation matrix to apply to the points
count	The number of points to transform
points	A pointer to an array of 2D points to transform
points_return	A pointer to an array in which to store the transformed points

Returns

Zero if successful; otherwise, one of the following:

PEXBadHomoCoord - one or more of the transformed points has a homogeneous coordinate of 0.

Description

This function applies the specified homogeneous transformation matrix to the list of points. In applying the transformation, the points are first converted to homogeneous points by assigning them a homogeneous coordinate of 1. The transformation is then applied:

P' = TxP

Where P is the point, treated as a column vector, and T is the transformation matrix. The points are then mapped to 2D by dividing their first three coordinates by the computed homogeneous coordinate.

If the function returns unsuccessfully, all points other than those with a homogeneous coordinate of 0 will be transformed and returned.

If the return array is the same as the input array, the function will overwrite the input values with the transformed values.

Errors

None

See Also

PEXTransformPoints, **PEXTransformPoints4D**, **PEXTransformPoints2DH**

PEXTransformPoints4D

Synopsis

void PEXTransformPoints4D(PEXMatrix *transform*, int *count*, PEXCoord4D **points*, PEXCoord4D **points_return*)

Arguments

transform	The transformation matrix to apply to the points
count	The number of points to transform
points	A pointer to an array of 4D points to transform
points_return	A pointer to an array in which to store the transformed points

Returns

None

Description

This function applies the specified homogeneous transformation matrix to the list of 3D homogeneous points (P = x, y, z, w). The transformation is applied:

$$P' = TxP$$

Where P is the point, treated as a column vector, and T is the transformation matrix. The function returns P' = (x', y', z', w').

If the return array is the same as the input array, the function will overwrite the input values with the transformed values.

Errors

None

See Also

PEXTransformPoints, **PEXTransformPoints2D**, **PEXTransformPoints2DH**

PEXTransformPoints2DH

Synopsis

void PEXTransformPoints2DH(PEXMatrix3x3 *transform*, int *count*, PEXCoord **points*, PEXCoord **points_return*)

Arguments

transform	The transformation matrix to apply to the points
count	The number of points to transform
points	A pointer to an array of 2D homogeneous points to transform
points_return	A pointer to an array in which to store the transformed points

Returns

None

Description

This function applies the specified homogeneous transformation matrix to the list of 2D homogeneous points (P = x, y, w). The transformation is applied:

P' = TxP

Where P is the point, treated as a column vector, and T is the transformation matrix. The function returns P' = (x', y', w').

If the return array is the same as the input array, the function will overwrite the input values with the transformed values.

Errors

None

See Also

PEXTransformPoints, **PEXTransformPoints2D**, **PEXTransformPoints4D**

PEXTransformVectors

Synopsis

void PEXTransformVectors(PEXMatrix *transform*, int *count*, PEXVector **vectors*, PEXVector **vectors_return*)

Arguments

transform	The transformation matrix to apply to the vectors
count	The number of vectors to transform
vectors	A pointer to the array of 3D vectors to transform
vectors_return	A pointer to an array in which to store the transformed vectors

Returns

None

Description

This function applies the upper 3x3 submatrix of the specified transformation matrix to the list of 3D vectors. The transformation is applied:

V’ = T’xV

Where V is the vector, treated as a column vector, and T’ is the upper 3x3 sub-matrix of "transform."

If the return array is the same as the input array, the function will overwrite the input values with the transformed values.

Errors

None

See Also

PEXTransformVectors2D, **PEXNormalizeVectors**

PEXTransformVectors2D

Synopsis

void PEXTransformVectors2D(PEXMatrix3x3 *transform*, int *count*, PEXVector2D **vectors*, PEXVector2D **vectors_return*)

Arguments

transform	The transformation matrix to apply to the vectors
count	The number of vectors to transform
vectors	A pointer to the array of 2D vectors to transform
vectors_return	A pointer to an array in which to store the transformed vectors

Returns

None

Description

This function applies the upper 2x2 submatrix of the specified transformation matrix to the list of 2D vectors. The transformation is applied:

$$V' = T' x V$$

Where V is the vector, treated as a column vector, and T' is the upper 2x2 sub-matrix of "transform."

If the return array is the same as the input array, the function will overwrite the input values with the transformed values.

Errors

None

See Also

PEXTransformVectors, **PEXNormalizeVectors2D**

PEXNormalizeVectors

Synopsis

int PEXNormalizeVectors(int *count*, PEXVector **vectors*, PEXVector **vectors_return*)

Arguments

count	The number of vectors to normalize
vectors	A pointer to the array of vectors to normalize
vectors_return	A pointer to an array in which to store the normalized vectors

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector - One of the vectors has zero magnitude.

Description

This function normalizes each vector in the specified list of 3D vectors. An error is returned if any vector in the list has a magnitude of zero. All non-zero vectors in the list are still normalized, however.

If the return array is the same as the input array, the function will overwrite the input values with the normalized values.

Errors

None

See Also

PEXTransformVectors, **PEXNormalizeVectors2D**

PEXNormalizeVectors2D

Synopsis

int PEXNormalizeVectors2D(int *count*, PEXVector2D **vectors*, PEXVector2D **vectors_return*)

Arguments

count	The number of vectors to normalize
vectors	A pointer to the array of vectors to normalize
vectors_return	A pointer to an array in which to store the normalized vectors

Returns

Zero if successful; otherwise, one of the following:

PEXBadVector - One of the vectors has zero magnitude.

Description

This function normalizes each vector in the specified list of 2D vectors. An error is returned if any vector in the list has a magnitude of zero. All non-zero vectors in the list are still normalized, however.

If the return array is the same as the input array, the function will overwrite the input values with the normalized values.

Errors

None

See Also

PEXTransformVectors2D, **PEXNormalizeVectors**

PEXNPCToXCTransform

Synopsis

int PEXNPCToXCTransform(PEXNPCSubVolume **npc_sub_volume*, PEXDeviceCoord **viewport*,
unsigned int *window_height*, PEXMatrix *transform_return*)

Arguments

npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource
viewport	An array of two device coordinate points defining a viewport, typically that of a renderer resource. The first point in the array is the lower-left corner of the viewport; the second point is the upper-right.
window_height	The height of the drawable
transform_return	The returned transformation

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)
PEXBadSubVolume - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)

Description

This function computes the transformation matrix to map an NPC point to a drawable coordinate (XC), using the specified NPC subvolume, DC viewport, and drawable height. The returned transformation matrix first applies the subvolume-to-viewport transformation, then transforms the x and y coordinates of the resulting points to drawable coordinates, leaving the z coordinate in DC.

When specifying NPC and DC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXNPCToXCTransform2D

PEXNPCToXCTransform2D

Synopsis

int PEXNPCToXCTransform2D(PEXNPCSubVolume *npc_sub_volume*, PEXDeviceCoord2D *viewport*, unsigned int *window_height*, PEXMatrix3x3 *transform_return*)

Arguments

npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource
viewport	An array of two device coordinate points defining a viewport, typically that of a renderer resource. The first point in the array is the lower-left corner of the viewport; the second point is the upper-right.
window_height	The height of the drawable
transform_return	The returned transformation

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)
PEXBadSubVolume - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)

Description

This function computes the 2D transformation matrix to map a 2D NPC point to a 2D drawable coordinate (XC), using the specified NPC subvolume, DC viewport, and drawable height. The returned transformation matrix first applies the subvolume-to-viewport transformation, then transform the resulting points to drawable coordinates.

When specifying NPC and DC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXNPCToXCTransform

PEXXCToNPCTransform

Synopsis

int PEXXCToNPCTransform(PEXNPCSubVolume **npc_sub_volume*, PEXDeviceCoord **viewport*, unsigned int *window_height*, PEXMatrix *transform_return*)

Arguments

npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource
viewport	An array of two device coordinate points defining a viewport, typically that of a renderer resource. The first point in the array is the lower-left corner of the viewport; the second point is the upper-right.
window_height	The height of the drawable
transform_return	The returned transformation

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)
PEXBadSubVolume - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)

Description

This function computes a transformation matrix to map a drawable point (XC) to NPC coordinates, using the specified NPC subvolume, DC viewport, and drawable height. The returned transformation matrix first transforms the x and y coordinates of points to device coordinates (DC), leaving the z coordinate unmodified. It then applies the viewport-to-subvolume transformation to all coordinates of the resulting points, producing 3D NPC points.

When specifying NPC and DC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXXCToNPCTransform2D

PEXXCToNPCTransform2D

Synopsis

int PEXXCToNPCTransform2D(PEXNPCSubVolume **npc_sub_volume*, PEXDeviceCoord2D **viewport*, unsigned int *window_height*, PEXMatrix3x3 *transform_return*)

Arguments

npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource
viewport	An array of two device coordinate points defining a viewport, typically that of a renderer resource. The first point in the array is the lower-left corner of the viewport; the second point is the upper-right.
window_height	The height of the drawable
transform_return	The returned transformation

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)
PEXBadSubVolume - (xmin >= xmax, or ymin >= ymax, or zmin > zmax)

Description

This function computes the 2D transformation matrix to map a drawable point (XC) to NPC coordinates, using the specified NPC subvolume, DC viewport, and drawable height. The returned transformation matrix first transforms the x and y coordinates of drawable points to device coordinates (DC), then applies the 2D components of the viewport-to-subvolume transformation, producing 2D NPC points.

When specifying NPC and DC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXXCToNPCTransform

PEXMapXCToNPC

Synopsis

int PEXMapXCToNPC(int *point_count*, PEXDeviceCoord2D **points*, unsigned int *window_height*, double *z_dc*, PEXDeviceCoord **viewport*, PEXNPCSubVolume **npc_sub_volume*, int *view_count*, PEXViewEntry **views*, int **view_return*, int **count_return*, PEXCoord **points_return*)

Arguments

point_count	The number of points to transform
points	A pointer to an array of points to transform. The X and Y coordinates of these points are in drawable coordinates (XC). The Z coordinate is in device coordinates (DC).
window_height	The height of the drawable
z_dc	The z DC coordinate to assign the drawable points when converting them to DC.
viewport	An array of two device coordinate points defining a viewport, typically that of a renderer resource. The first point in the array is the lower-left corner of the viewport; the second point is the upper-right.
npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource
view_count	The number of views to search
views	The view entries to search for inclusion of the transformed points
view_return	Returns the view found to contain the most points
count_return	Returns the number of points contained in the returned view, or the number of points transformed if no views are specified.
points_return	Returns a pointer to an array in which to store the transformed points.

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport
PEXBadSubVolume

Description

This function maps a list of drawable coordinates (XC) to NPC, and searches a specified list of view entries to determine the view containing the computed NPC points.

The XC points are first transformed to DC, using the specified window height and assigning them the specified z DC value. They are then transformed to NPC by the viewport-to-subvolume transform implied by the specified viewport and NPC subvolume. The specified list of views is then searched, in order from 0 to the number of views minus 1, and the index of the first view containing all the NPC points is returned. If no view contains all the points, then the lowest-index view containing the most points is returned. In this case, only the points within the view are returned in "points_ret."

When determining the containing view, only the clipping limits of the view are considered, with no consideration given to the clipping flags or the viewing transforms.

If no views are specified, the XC points are simply transformed to NPC points and returned. The value of the returned view is undefined in this case.

The viewport-to-subvolume transformation maps to NPC the largest region of the specified viewport that has the same aspect ratio as the NPC subvolume and is anchored at the back lower-left of the viewport (the corner of the viewport with the minimum X, Y and Z coordinates). Points that lie outside this region of the viewport are not transformed.

When specifying NPC and DC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXMapXCToNPC2D, **PEXNPCToXCTransform** **PEXXCToNPCTransform**

PEXMapXCToNPC2D

Synopsis

int PEXMapXCToNPC2D(int *point_count*, PEXDeviceCoord2D **points*, unsigned int *window_height*, PEXDeviceCoord2D **viewport*, PEXNPCSubVolume **npc_sub_volume*, int *view_count*, PEXViewEntry **views*, int **view_return*, int **count_return*, PEXCoord2D **points_return*)

Arguments

point_count	The number of points to transform
points	A pointer to an array of drawable-coordinate (XC) points to transform.
window_height	The height of the drawable
viewport	An array of two device coordinate points defining a viewport, typically that of a renderer resource. The first point in the array is the lower-left corner of the viewport; the second point is the upper-right.
npc_sub_volume	A pointer to an NPC subvolume, typically that of a renderer resource
view_count	The number of views to search
views	The view entries to search for inclusion of the transformed points
view_return	The view found to contain the most points
count_return	Returns the number of points contained in the returned view, or the number of points transformed if no views are specified.
points_return	A pointer to an array in which to store the transformed points.

Returns

Zero if successful; otherwise, one of the following:

PEXBadViewport
PEXBadSubVolume

Description

This function maps a list of drawable coordinates (XC) to NPC, and searches a specified list of view entries to determine the view containing the computed NPC points.

The XC points are first transformed to 2D DC, using the specified window height, then transformed to 2D NPC by the viewport-to-subvolume transform implied by the specified viewport and NPC subvolume. The specified list of views is then searched, in order from 0 to the number of views minus 1, and the index of the first view containing all the NPC points is returned. If no view contains all the points, then the lowest-index view containing the most points is returned. In this case, only the points within the view are returned.

When determining the containing view, only the x-y clipping limits of the view are considered, with no consideration given to the front and back clipping limits, the clipping flags, or the viewing transforms.

If no views are specified, the XC points are simply transformed to NPC points and returned. The value of the returned view is undefined in this case.

The viewport-to-subvolume transformation maps to NPC the largest region of the specified viewport that has the same aspect ratio as the NPC subvolume and is anchored at the back lower-left of the viewport (the corner of the viewport with the minimum X, Y and Z coordinates). Points that lie outside this region of the viewport are not transformed.

When specifying NPC and DC, the X, Y and Z limits must be as follows:

xmin < xmax , ymin < ymax , zmin <= zmax

Errors

None

See Also

PEXMapXCToNPC, **PEXNPCToXCTransform2D** **PEXXCToNPCTransform2D**

PEXInvertMatrix

Synopsis

int PEXInvertMatrix(PEXMatrix *transform*, PEXMatrix *transform_return*)

Arguments

transform	The transformation matrix to invert
transform_return	The inverse transformation

Returns

Zero if successful; non-zero if unsuccessful.

Description

This function computes the inverse of a transformation matrix.

An unsuccessful status is returned if the matrix is non-invertible.

The two arguments may be the same variable, in which case the inversion is performed in-place, overwriting the original transform.

Errors

None

See Also

PEXInvertMatrix2D

PEXInvertMatrix2D

Synopsis

int PEXInvertMatrix2D(PEXMatrix3x3 *transform*, PEXMatrix3x3 *transform_return*)

Arguments

transform	The transformation matrix to invert
transform_return	The inverse transformation

Returns

Zero if successful; non-zero if unsuccessful.

Description

This function computes the inverse of a 2D transformation matrix.

An unsuccessful status is returned if the matrix is non-invertible.

The two arguments may be the same variable, in which case the inversion is performed in-place, overwriting the original transform.

Errors

None

See Also

PEXInvertMatrix

PEXIdentityMatrix

Synopsis

void PEXIdentityMatrix(PEXMatrix *transform_return*)

Arguments

transform_return The returned identity matrix

Description

This function returns an identity matrix.

Errors

None

See Also

PEXIdentityMatrix2D

Synopsis

void PEXIdentityMatrix2D(PEXMatrix3x3 *transform_return*)

Arguments

transform_return The returned identity matrix

Description

This function returns a 2D identity matrix.

Errors

None

See Also

16.4. Utilities for Computing Geometric Normals

These utilities compute the geometric normal vectors of the complex PEX output primitives. Their input parameters are identical to the parameters of the corresponding output command function. The facet-data return parameter is filled in with the computed geometric normal of each facet.

PEXGeoNormFillArea
PEXGeoNormFillAreaSet
PEXGeoNormQuadrilateralMesh
PEXGeoNormSetOfFillAreaSets
PEXGeoNormTriangleStrip

16.4.1. Common Data Structures

Below are the data structures used that are common to more than one function described in this section.

```
/* coordinates */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXCoord;

      typedef struct {
          float               x;
          float               y;
      } PEXCoord2D;

      typedef struct {
          unsigned long       count;                  /* number of points */
          PEXCoord            *points;
      } PEXListOfCoord;       /* Pointer to an array of 3D points */

      typedef struct {
          unsigned long       count;                  /* number of points */
          PEXCoord2D          *points;
      } PEXListOfCoord2D;     /* Pointer to an array of 2D points */

/* vectors */

      typedef struct {
          float               x;
          float               y;
          float               z;
      } PEXVector;

/* facet data */

      typedef union {
          PEXColorIndexed     index;
          PEXColorRGB         rgb;
          PEXColorHSV         hsv;
          PEXColorHLS         hls;
          PEXColorCIE         cie;
          PEXColorRGB8        rgb8;
          PEXColorRGB16       rgb16;
          PEXVector           normal;
          PEXColorIndexedNormal index_normal;
```

```
    PEXColorRGBNormal   rgb_normal;
    PEXColorHSVNormal   hsv_normal;
    PEXColorHLSNormal   hls_normal;
    PEXColorCIENormal   cie_normal;
    PEXColorRGB8Normal  rgb8_normal;
    PEXColorRGB16Normal rgb16_normal;
} PEXFacetData;

typedef union {
    PEXColorIndexed     *index;
    PEXColorRGB         *rgb;
    PEXColorHSV         *hsv;
    PEXColorHLS         *hls;
    PEXColorCIE         *cie;
    PEXColorRGB8        *rgb8;
    PEXColorRGB16       *rgb16;
    PEXVector           *normal;
    PEXColorIndexedNormal *index_normal;
    PEXColorRGBNormal   *rgb_normal;
    PEXColorCIENormal   *cie_normal;
    PEXColorHSVNormal   *hsv_normal;
    PEXColorHLSNormal   *hls_normal;
    PEXColorRGB8Normal  *rgb8_normal;
    PEXColorRGB16Normal *rgb16_normal;
} PEXArrayOfFacetData;

typedef struct {
    PEXTableIndex       index;
    unsigned short      reserved;
} PEXColorIndexed;

typedef struct {
    float               red;
    float               green;
    float               blue;
} PEXColorRGB;

typedef struct {
    float               hue;
    float               saturation;
    float               value;
} PEXColorHSV;

typedef struct {
    float               hue;
    float               lightness;
    float               saturation;
} PEXColorHLS;

typedef struct {
    float               x;
    float               y;
    float               z;
} PEXColorCIE;
```

```
    typedef struct {
        unsigned char       red;
        unsigned char       green;
        unsigned char       blue;
        unsigned char       reserved;
    } PEXColorRGB8;

    typedef struct {
        unsigned short      red;
        unsigned short      green;
        unsigned short      blue;
        unsigned short      reserved;
    } PEXColorRGB16;

    typedef struct {
        PEXColorIndexed     index;
        PEXVector           normal;
    } PEXColorIndexedNormal;

    typedef struct {
        PEXColorRGB         rgb;
        PEXVector           normal;
    } PEXColorRGBNormal;

    typedef struct {
        PEXColorCIE         cie;
        PEXVector           normal;
    } PEXColorCIENormal;

    typedef struct {
        PEXColorHSV         hsv;
        PEXVector           normal;
    } PEXColorHSVNormal;

    typedef struct {
        PEXColorHLS         hls;
        PEXVector           normal;
    } PEXColorHLSNormal;

    typedef struct {
        PEXColorRGB8        rgb8;
        PEXVector           normal;
    } PEXColorRGB8Normal;

    typedef struct {
        PEXColorRGB16       rgb16;
        PEXVector           normal;
    } PEXColorRGB16Normal;

/* vertex data */

    typedef struct {
        unsigned long       count;                  /* number of vertices */
        PEXArrayOfVertex    vertices;               /* pointer to vertices */
    } PEXListOfVertex;
```

```
typedef union {
    PEXCoord                    *no_data;
    PEXVertexIndexed            *index;
    PEXVertexRGB                *rgb;
    PEXVertexHSV                *hsv;
    PEXVertexHLS                *hls;
    PEXVertexCIE                *cie;
    PEXVertexRGB8               *rgb8;
    PEXVertexRGB16              *rgb16;
    PEXVertexNormal             *normal;
    PEXVertexEdge               *edge;
    PEXVertexIndexedNormal      *index_normal;
    PEXVertexRGBNormal          *rgb_normal;
    PEXVertexHSVNormal          *hsv_normal;
    PEXVertexHLSNormal          *hls_normal;
    PEXVertexCIENormal          *cie_normal;
    PEXVertexRGB8Normal         *rgb8_normal;
    PEXVertexRGB16Normal        *rgb16_normal;
    PEXVertexIndexedEdge        *index_edge;
    PEXVertexRGBEdge            *rgb_edge;
    PEXVertexHSVEdge            *hsv_edge;
    PEXVertexHLSEdge            *hls_edge;
    PEXVertexCIEEdge            *cie_edge;
    PEXVertexRGB8Edge           *rgb8_edge;
    PEXVertexRGB16Edge          *rgb16_edge;
    PEXVertexNormalEdge         *normal_edge;
    PEXVertexIndexedNormalEdge  *index_normal_edge;
    PEXVertexRGBNormalEdge      *rgb_normal_edge;
    PEXVertexHSVNormalEdge      *hsv_normal_edge;
    PEXVertexHLSNormalEdge      *hls_normal_edge;
    PEXVertexCIENormalEdge      *cie_normal_edge;
    PEXVertexRGB8NormalEdge     *rgb8_normal_edge;
    PEXVertexRGB16NormalEdge    *rgb16_normal_edge;
} PEXArrayOfVertex;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
} PEXVertexIndexed;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
} PEXVertexRGB;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
} PEXVertexHSV;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
} PEXVertexHLS;
```

```
typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
} PEXVertexCIE;

typedef struct {
    PEXCoord            point;
    PEXColorRGB8        rgb8;
} PEXVertexRGB8;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
} PEXVertexRGB16;

typedef struct {
    PEXCoord            point;
    PEXVector           normal;
} PEXVertexNormal;

typedef struct {
    PEXCoord            point;
    unsigned int        edge;
} PEXVertexEdge;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
    PEXVector           normal;
} PEXVertexIndexedNormal;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
    PEXVector           normal;
} PEXVertexRGBNormal;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
    PEXVector           normal;
} PEXVertexHSVNormal;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
    PEXVector           normal;
} PEXVertexHLSNormal;

typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
    PEXVector           normal;
} PEXVertexCIENormal;
```

```
typedef struct {
    PEXCoord            point;
    PEXColorRGB8        rgb8;
    PEXVector           normal;
} PEXVertexRGB8Normal;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
    PEXVector           normal;
} PEXVertexRGB16Normal;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
    unsigned int        edge;
} PEXVertexIndexedEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
    unsigned int        edge;
} PEXVertexRGBEdge;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
    unsigned int        edge;
} PEXVertexHSVEdge;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
    unsigned int        edge;
} PEXVertexHLSEdge;

typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
    unsigned int        edge;
} PEXVertexCIEEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB8        rgb8;
    unsigned int        edge;
} PEXVertexRGB8Edge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
    unsigned int        edge;
} PEXVertexRGB16Edge;
```

```
typedef struct {
    PEXCoord            point;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexNormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorIndexed     index;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexIndexedNormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB         rgb;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexRGBNormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorHSV         hsv;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexHSVNormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorHLS         hls;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexHLSNormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorCIE         cie;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexCIENormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB8        rgb8;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexRGB8NormalEdge;

typedef struct {
    PEXCoord            point;
    PEXColorRGB16       rgb16;
    PEXVector           normal;
    unsigned int        edge;
} PEXVertexRGB16NormalEdge;
```

PEXGeoNormFillArea

Synopsis

int PEXGeoNormFillArea(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, PEXFacetData **facet_data*, unsigned int *count*, PEXArrayOfVertex *vertices*)

Arguments

facet_attributes	A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.
vertex_attributes	A mask indicating the vertex attributes provided.
color_type	The type of color data provided.
facet_data	A pointer to facet data. This function adds the geometric normal to this data.
count	The number of vertices.
vertices	An array of vertices defining the fill area.

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive - A normal cannot be computed because the fill area is degenerate or because all vertices are colinear.

Description

This function computes the geometric normal of a fill area and stores it in the specified facet data.

The normal is computed by finding the first three non-colinear points in the specified vertices, forming two vectors from those points, one from the first point to the second point and one from the first point to the third point, and computing the cross product of those two vectors. The geometric normal is the normalized cross product.

The three points, A, B, and C are selected as follows. Point A is the first point in the list of vertices. Point B is the next point in the list that is not coincident with A. Point C is the next point in the list that is not colinear with A and B. If it is not possible to find three such points, the functions returns unsuccessfully.

If the facet attributes does not contain the bit **PEXGANormal**, the geometric normal is not computed. However, the function still returns successfully.

Errors

None

See Also

PEXGeoNormFillAreaSet

Synopsis

int PEXGeoNormFillAreaSet(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, unsigned int *count*, PEXFacetData **facet_data*, PEXListOfVertex **vertex_lists*)

Arguments

facet_attributes	A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.
vertex_attributes	A mask indicating the vertex attributes provided.
color_type	The type of color data provided.
count	The number of fill areas in the set.
facet_data	An array of facet data. This function adds the geometric normal to this 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:

PEXBadPrimitive - *A normal cannot be computed because all fill areas in the set are degenerate or because the vertices of all the fill areas are colinear.*

Description

This function computes the geometric normal of a fill area set and stores it in the specified facet data.

The normal is computed by finding the first three non-colinear points in a fill area of the set, beginning with the first fill area and searching until three such points are found in a single fill area. Two vectors are formed from these points: one vector from the first point to the second point, and one vector from the first point to the third point. The geometric normal returned is the normalized cross product of those two vectors.

The three points, A, B, and C are selected as follows. Point A is the first point in the first list of vertices. Point B is the next point in that same list that is not coincident with A. Point C is the next point in that same list that is not colinear with A and B. If it is not possible to find three such points in the first list, then the rest of the lists are searched in order to select three appropriate points from a single list. If it is still not possible to find three such points in any list, the functions returns unsuccessfully.

If the facet attributes does not contain the bit **PEXGANormal**, the geometric normal is not computed. However, the function still returns successfully.

Errors

None

See Also

PEXGeoNormQuadrilateralMesh

Synopsis

int PEXGeoNormQuadrilateralMesh(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, PEXArrayOfFacetData *facet_data*, unsigned int *col_count*, unsigned int *row_count*, PEXArrayOfVertex *vertices*)

Arguments

facet_attributes	A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.
vertex_attributes	A mask indicating the vertex attributes provided.
color_type	The type of color data provided.
facet_data	An array of facet data. This function adds the geometric normals to this 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:

PEXBadPrimitive - *A normal cannot be computed for one or more quadrilaterals in the mesh.*

Description

This function computes the geometric normals of a quadrilateral mesh and stores them in the specified facet data.

The geometric normal of each quadrilateral is computed by forming two vectors from two of its sides, and computing the cross product of those two vectors. The geometric normal is the normalized cross product:

Ng = (V1 x V2) / |V1 X V2|

Given the quadrilateral composed of four vertices, Pi,j, where i indicates the row of the point and j its column, the first vector, V1, is from Pi,j to Pi+1,j+1. The second vector, V2, is from Pi+1,j to Pi,j+1.

If the facet attributes does not contain the bit **PEXGANormal**, the geometric normal is not computed. However, the function still returns successfully.

A geometric normal is computed for all quadrilaterals where it is possible to compute one, even if a normal cannot be computed for some other quadrilaterals. An error is returned if a normal cannot be computed for one or more of the quadrilaterals in the mesh.

Errors

None

See Also

PEXGeoNormSetOfFillAreaSets

Synopsis

int PEXGeoNormSetOfFillAreaSets(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, unsigned int *set_count*, PEXArrayOfFacetData *facet_data*, unsigned int *vertex_count*, PEXArrayOfVertex *vertices*, unsigned int *index_count*, PEXConnectivityData **connectivity*)

Arguments

facet_attributes	A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.
vertex_attributes	A mask indicating the vertex attributes provided.
color_type	The type of color data provided.
set_count	The number of fill area sets.
facet_data	An array of facet data. This function adds the geometric normals to this data.
vertex_count	The number of vertices.
vertices	An array of vertices.
index_count	The number of vertex connectivity indices.
connectivity	A pointer to the list of contour connectivity data.

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive - A normal cannot be computed for a fill area set because all its fill areas are degenerate or because all the vertices of all fill areas in the set are colinear.

Description

This function computes the geometric normals of set of fill area sets primitive and stores them in the specified facet data.

The normals are computed by finding the first three non-colinear points in each fill area set, beginning with the first fill area of each set and searching until three such points are found in a single fill area. Two vectors are formed from these points: one vector from the first point to the second point, and one vector from the first point to the third point. The geometric normal returned is the normalized cross product of these two vectors.

The three points for each fill area set are selected as described for **PEXGeoNormFillAreaSet**.

If the facet attributes does not contain the bit **PEXGANormal**, the geometric normal is not computed. However, the function still returns successfully.

A geometric normal is computed for all fill area sets where it is possible to compute one, even if a normal cannot be computed for some other fill area sets. The function returns unsuccessfully if a normal cannot be computed for one or more of the fill area sets.

Data Structures

```
typedef struct {
    unsigned short      count;                  /* number of lists */
    PEXListOfUShort     *lists;
} PEXConnectivityData;

typedef struct {
    unsigned short      count;                  /* number of shorts */
```

```
    unsigned short      *shorts;
} PEXListOfUShort;
```

See also the *Common Data Structures* section.

Errors

None

See Also

PEXGeoNormTriangleStrip

Synopsis

int PEXGeoNormTriangleStrip(unsigned int *facet_attributes*, unsigned int *vertex_attributes*, int *color_type*, PEXArrayOfFacetData *facet_data*, unsigned int *count*, PEXArrayOfVertex *vertices*)

Arguments

facet_attributes A mask indicating the facet attributes provided. It should contain the bit **PEXGANormal**.

vertex_attributes A mask indicating the vertex attributes provided.

color_type The type of color data provided.

facet_data An array of facet data. This function adds the geometric normals to this data.

count The number of vertices.

vertices An array of vertices defining the triangle strip.

Returns

Zero if successful; otherwise, one of the following:

PEXBadPrimitive - *A normal cannot be computed for one or more triangles in the strip.*

Description

This function computes the geometric normals of a triangle strip and stores them in the specified facet data.

The geometric normal of each triangle is computed by forming two vectors from two of its sides, and computing the cross product of those two vectors. The geometric normal is the normalized cross product:

Ng = (V1 x V2) / |V1 X V2|

For the first, third, and subsequent odd-numbered triangles, the first vector (V1) is from the first point (Pi) of the triangle to the second point (Pi+1), and the second vector (V2) is from the first point of the triangle to the third point (Pi+2). For the second, fourth, and subsequent even-numbered triangles, the first vector is from the first point (Pi) of the triangle to the third point (Pi+2), and the second vector is from the first point of the triangle to the second point (Pi+1).

If the facet attributes does not contain the bit **PEXGANormal**, the geometric normal is not computed. However, the function still returns successfully.

A geometric normal is computed for all triangles where it is possible to compute one, even if a normal cannot be computed for some other triangles. An error is returned if a normal cannot be computed for one or more of the triangles in the strip.

Errors

None

See Also

Appendix A

PEXlib Programming for both 5.0 and 5.1

Introduction

A PEXlib program written to use only functionality specified in PEX 5.0 will work with a PEX server that supports either PEX 5.0 or PEX 5.1 protocol. However, if a PEXlib program uses functionality present only in PEX 5.1, the program will likely fail on a PEX 5.0 server.

This addendum gathers the differences between PEX 5.0 and PEX 5.1 in one place so the programmer can understand and account for the changes. The programmer may wish to take advantage of PEX 5.1 functions, if present, and perform some alternate action if not.

Determining the Protocol Version

The first task is for the programmer to determine the protocol version that the server supports, so the program may behave accordingly. This is accomplished with the **PEXGetExtensionInfo** function:

```
int                     status;
PEXExtensionInfo        *pexinfo;
char                    errstr[PEXErrorStringLength];
status = PEXInitialize( display, PEXErrorStringLength, errstr );
if (status != 0) {
    fprintf( stderr, "%s0, errstr );
    exit(0);
}
pexinfo = PEXGetExtensionInfo( display );

/* The server is a PEX 5.1 server if (pexinfo->major_version == 5) */
/* and (pexinfo->minor_version == 1). */
```

Events

PEX 5.1 defines a new X Event, **PEXMaxHitsReached**. Some 5.1 programs that use **PEXPickAll** may wish to watch for this event to decide if additional pick passes are necessary. This event will not be sent by a PEX 5.0 server.

Enumerated Type Information

PEXETHLHSRMode - PEX 5.1 defines the new registered value **PEXHLHSRZBufferID**
PEXETPickOneMethod - New enum type for 5.1.
PEXETPickAllMethod - New enum type for 5.1.
PEXETEscape - New enum type for 5.1

Applications should not expect to find the **PEXHLHSRZBufferID** standard value when querying the HLHSR mode enum type from a 5.0 server and should therefore not attempt to use it. Applications should not attempt to query the other three types from a PEX 5.0 server.

Implementation Dependent Constants

PEXIDMaxHitsEventSupported - New for PEX 5.1

Applications should not attempt to query this value using the **PEXGetImpDepConstants** function from a PEX 5.0 server.

General Functions

The following PEXlib functions are new for PEX 5.1 and should not be sent to a PEX 5.0 server.

PEXMatchRenderingTargets
PEXEscape
PEXEscapeWithReply
PEXSetEchoColor

Output Commands

The **PEXNoop** output command function must not be sent to a PEX 5.0 server or stored in a structure network that will be rendered by a PEX 5.0 server.

Renderer Functions

The following PEXlib functions are new for PEX 5.1 and should not be sent to a PEX 5.0 server.

PEXRenderElements
PEXAccumulateState

Renderer Dynamics

The following dynamics bits cannot be queried from a PEX 5.0 server.

PEXRDNPickNameSet
PEXRDNPickNameSetContents
PEXRDAEchoMode

Renderer Attributes

The following renderer attributes should not be used with a PEX 5.0 server.

PEXRAPickIncl
PEXRAPickExcl
PEXRAPickStartPath
PEXRABackgroundColor
PEXRAClearImage
PEXRAClearZ
PEXRAEchoMode

Renderer Picking Functions

The following PEXlib functions are new for PEX 5.1 and should not be sent to a PEX 5.0 server.

PEXPickOne
PEXBeginPickOne
PEXEndPickOne
PEXPickAll
PEXBeginPickAll
PEXEndPickAll

Table of Contents